Discipline of Computing and Information Technology
Semester 1, 2022 - SENG1120/6120

Assignment 2

Due using the Blackboard Assignment submission facility:
11:59PM – Sunday May 15th, 2022

Specification Version 2.0.0

NOTE: *The important information about submission and code specifics at the end of this assignment specification.*

INTRODUCTION

In lectures we have discussed the use of `templates` to provide the compiler with blueprints for functions and classes. These are used by the compiler to produce code that implements functions and/or classes that are independent of the type(s) with which you will use them in your program code. In *Assignment 2*, that means you will create a `Stack` data structure, into which you can put any data type, then demonstrate that `Stack` by using it as the basis for a simple version of **The Towers of Hanoi**.

PROBLEM DESCRIPTION

Your task in this Assignment is to implement a traditional game - the **Towers of Hanoi**. The Towers of Hanoi is a **mathematical puzzle** consisting of three rods, and several disks of different sizes that can slide onto any rod. The puzzle starts with the disks neatly stacked in order of size on the left rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to the right rod, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and placing it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

For complete information about it, and to play it, visit the following websites:
https://www.mathsisfun.com/games/towerofhanoi.html
http://en.wikipedia.org/wiki/Tower_of_Hanoi

ASSIGNMENT TASK

You will take your `Node` and `LinkedList` classes from Assignment 1 and produce a:

1. class template `Node`, which is then used as a component in the construction of
2. class template `LinkedList`, which is then used as a component in the construction of
3. class template `LStack`, which is then used as a component in the construction of
4. class `TowerHanoi`, which implements the functionality of the game.

`Node` and `LinkedList` should be dynamic. You will provide a complete `LinkedList`, with all standard `LinkedList` functions present and functional (ie: *do not remove unused functions*); a description of the minimum required functions can be found later in this document.

Public functions from `TowerHanoi` will be used by the main program, `TowerHanoiDemo`, ***which is provided to you***. `TowerHanoiDemo` will interface with the user to get the initial configuration and then start the game. The game will require the initialisation of several instances of `Disc`, ***which is provided to you*** and must be used in this assignment. The class `Disc` represents the discs used in the game and contain two member variables – one used for printing, and another for the disc's size.

We are also providing a `makefile`, which like all other provided files, should not be modified!

Having created and confirmed the correct operation of class templates, you will implement **The Tower of Hanoi** as follows:

Upon initialisation, `TowerHanoi` will create three instances of `LStack<Disc>`. In addition, several instances of `Disc` must be created at the start, and will depend on how many discs are used (*input at runtime*).

Once initialised, the state of the game is output by the `TowerHanoiDemo` class (*assuming 5 discs are being used*) and should produce the same output as below:

```
      X
     XXX
    XXXXX
   XXXXXXX
  XXXXXXXXX
---------------------------------
      1         2         3
```

The game will then ask the player to input a rod to remove a disc from and a rod to place a disc onto (*this happens in* `TowerHanoiDemo`) – you will need to verify if this is a valid move:

```
Please input the rod to remove from (1, 2 or 3): 1
Please input the rod to place onto (1, 2 or 3): 3
```

Suppose the player moves a disc from rod 1 to rod 3 as the first move; when the game is printed the next time, the output will be:

```
     XXX
    XXXXX
   XXXXXXX
  XXXXXXXXX                X
---------------------------------
      1         2         3
```

These last steps will *repeat and continue* (*this happens inside* `TowerHanoiDemo`) until all five discs are found correctly moved to rod 3 – once you check for this state, the game should display:

```
                          X
                         XXX
                        XXXXX
                       XXXXXXX
                      XXXXXXXXX
---------------------------------
      1         2         3

The game has ended.
You required 31 moves to win.
```

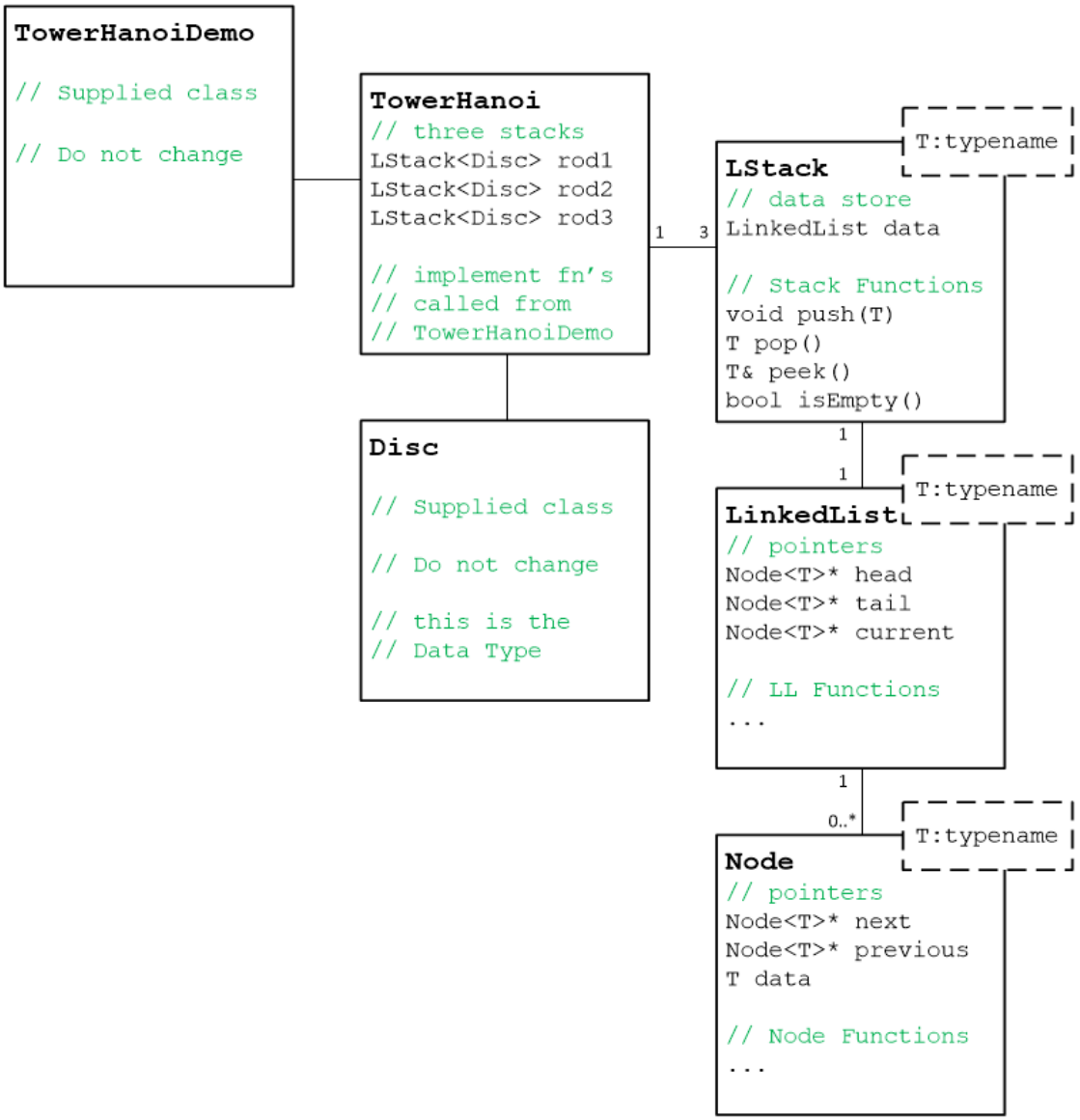
The program now ends.

CLASS RELATIONSHIPS

As above, you are provided the `TowerHanoiDemo` and `Disc` classes. Other classes will be created by you, as shown in the guide below – note, not all required class members are shown; you will need to discover what functions `TowerHanoi` needs to interact with `TowerHanoiDemo` (*hint, analyse* `TowerHanoiDemo` *and look for when it creates and interacts with the* `TowerHanoi` *game class*).

`LStack`, `LinkedList` and `Node` are required to be templates. `TowerHanoi` will be a normal, concrete class.

All source must compile with the provide C++ 98 `makefile`.



DATA STRUCTURE REQUIREMENTS

`Stack`

As discussed in lecture, your `LinkedList` class will become a private member of `LStack`. In this class you will provide implementation for the **Stack Public Interface**, below:

```
void push ( T item );
T pop ( );
T& peek ( );
boolean isEmpty ( );
```

Only these four functions are allowed to access/communicate with the `LinkedList` (*which was called* `data` *in Lectures*) – all other functions and manipulations of your Stack contents, must be done ONLY via these four Functions; you may not use any other *public* or *private* helper functions to access `LinkedList` from the `LStack` class.

From here you will obviously have to implement a constructor and destructor as well.

`LinkedList`

You are required to implement a fully functional generic version of your `LinkedList`; this will include the following Public Interface (1. I am OK if you are using `snake_case` or `camelCase`, 2. I am OK with minor variations of the function names, and, 3. your parameter names may be different; some will require `const`):

```
// add
void addToHead(T& item);
void addToTail(T& item);
void addToCurrent(T& item);

// get
T& getFromHead();     // optional
T& getFromTail();     // optional
T& getFromCurrent();

// remove
T removeFromHead();
T removeFromTail();
T removeFromCurrent();

// current manipulators
void start();
void end();
void forward();
void back();

// statistics
int size();
```

`LinkedList` will require a supporting `Node` object as well; no overloaded operators are required, however you will have to implement a constructor and destructor.

SENG6120/BONUS TASK

(1.0 mark) For students in SENG6120, there is an extra requirement:

1. Instead of having the three rods stored in an array, or as separate variables, the implementation of TowerHanoi must use the C++ STL container `List<LStack>` and populate the list with the 3 rods.

For SENG1120 students who want to be challenged more, the above requirement becomes a bonus question, also worth 1.0 marks; *however* you can still only score a MAXIMUM of 10.0/10.0.

SUBMISSION

Make sure your code works with the files supplied, and DO NOT change them. For marking, we will add the main file and the `Disc` class to the project and compile everything using the `makefile`, together with your own files. **If it does not compile or run, your mark will be zero (as we are unable to test and validate your implementation).**

Your submission should be made using the Assignments section of the course Blackboard site. **Incorrectly submitted assignments will not be marked.** You should provide the `.h/.hpp` and `.cpp` files related to the `LStack`, `LinkedList` and `Node` classes, only. Also, if necessary, provide a `readme.txt` file containing instructions or comments for the marker. Each program file should have a proper header comment section including your name, course and student number; and your code should be properly documented.

Remember that your code will conform to C++98 standards, and should compile and run correctly using Cygwin, gcc and the supplied makefile. There should be no segmentation faults or memory leaks during or after the execution of the program.

Compress all your files into a *single .zip file*, using your **student number** as the filename (*do not use .rar, .7z, .gz, or any other compressed format*). For example, if your student number is **`c9876543`**, you would name your submission:

`c9876543.zip`

If you have attempted the Bonus Requirement (*or you are a 6120 student*), please include a blank text file in the same folder as your source files, simply called **`Bonus.txt`** – this is to make it clear to the marker that you are attempting this.

Submit by selecting the **Assignment 2** link that will be found in the **Assignments** section on **Canvas**.

Late submissions are subject to the rules specified in the Course Outline.

This assignment is worth 10.0 marks of your final result for the course.

Dan and Alex
2022-04-30
v.2.0.0