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

Assignment 2

Due using the Canvas Assignment submission facility:
11:59PM – September 25th, 2022

Specification version 2.0.1

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 suitable for the type(s) used in your program code.

Sorting is one of the pillars of computer science, and is required for several tasks, e.g. database searching, optimisation, simulations, computer graphics, etc. Additionally it is a common part of technical interviews in many large software companies:

https://www.geeksforgeeks.org/top-10-algorithms-in-interview-questions/#algo4

PROBLEM DESCRIPTION

In this task you are required to **implement the support classes** for the algorithm **DanSort**, which is a made-up sorting algorithm, of unknown complexity (*invented by* Alex, *but named after* Dan). The algorithm uses a divide and conquer approach with stacks to store sorted sub-sequences of elements. **You will not have to implement this Algorithm.**

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 by the supplied Demo files.
4. class template `LQueue`, which is then used by the supplied Demo files.

All `Node` instances should be dynamic, and `LinkedList` should be a private member of your `LQueue` and `LStack` implementations. You will provide a complete `LinkedList`, with all standard `LinkedList` functions present and functional (ie: *you will provide valid implementation for all public interface functions - see next pages*); a description of the minimum required functions can be found later in this document.

You will be supplied with two main programs, `DanSortDemoInt` and `DanSortDemoString`. Both will interface independently with your `LQueue` and `LStack` implementation.

You are NOT ALLOWED to change the demo files or the `makefile`. Also note that `LStack`, `LQueue`, `LinkedList` and `Node` must work with **both demo files**. **You are not allowed to produce two versions of the files to be submitted.**

MAKEFILE NOTES

You will note that two makefiles have been provided – one that works with `DanSortDemoInt` and the other with `DanSortDemoString` – named `makefileInt` and `makefileString`. To properly test with both demo files you will need to rename each of these to simply `makefile` when you wish to test.

DANSORT ALGORITHM NOTES

Here is the pseudo-code of how our made-up DanSort Algorithm works.

Consider *k* unsorted elements stored in a queue, and *t* empty stacks (*files named* `DanSortDemoInt` *and* `DanSortDemoString` *will be provided to you as a .cpp files*):

```
// Emptying the queue and populating the stacks
for i=1 to k
  Remove the front item in the queue
  Place the item on the first stack whose top() is either null or greater than the
  item
endFor
** At this point, each stack will have a small, ordered sequence of values (or will
be empty), and the queue will be empty. **

// Emptying the stacks in an ordered fashion and re-populating the queue
for i=1 to k
  Remove the smallest item among all the stacks  // of course, empty stacks should be
                                                 //  skipped. Note that you only have
                                                 //     to compare the top() value of
                                                 //   each stack to find the minimum.
  Enqueue the item in the queue
endFor
** At this point, all stacks will be empty and the queue will be sorted. **
```

DATA STRUCTURE REQUIREMENTS

`LQueue`

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

```
void enqueue ( T item );
T dequeue ( );
T& front ( );
boolean isEmpty ( );
int size ( );
```

Only these five functions are allowed to access/communicate with the `LinkedList` – all other functions and manipulations of your Queue 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 `LQueue` class.

From here you will obviously have to implement a constructor and destructor as well, plus an output operator (`operator <<`), and correctly define your private members. You will also have to const appropriate functions

`LStack`

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` – 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, plus an output operator (`operator <<`), and correctly define your private members.

`LinkedList`

You are required to implement a fully functional generic version of your `LinkedList`; this will include the following Public Interface (Note 1. *I am OK if you are using* `snake_case` or `camelCase`, Note 2. *I am OK with minor variations of the function names, and*, Note 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 ( );
T& getFromTail ( );
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, and correctly define your private members.

SUBMISSION

Make sure your code works with the files supplied, and DO NOT change them. For marking, we will add the main file `DanSortDemoInt.cpp` or `DanSortDemoString.cpp` to the project and compile it 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 Canvas site. **Incorrectly submitted assignments will not be marked.** You should provide the `.h/.hpp` and `.cpp` files related to the `LQueue`, `LStack`, `LinkedList` and `Node` classes. Also, if necessary, provide a `readme.txt` file containing any instructions for the marker. Each program file should have a proper header 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/s. 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`

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.

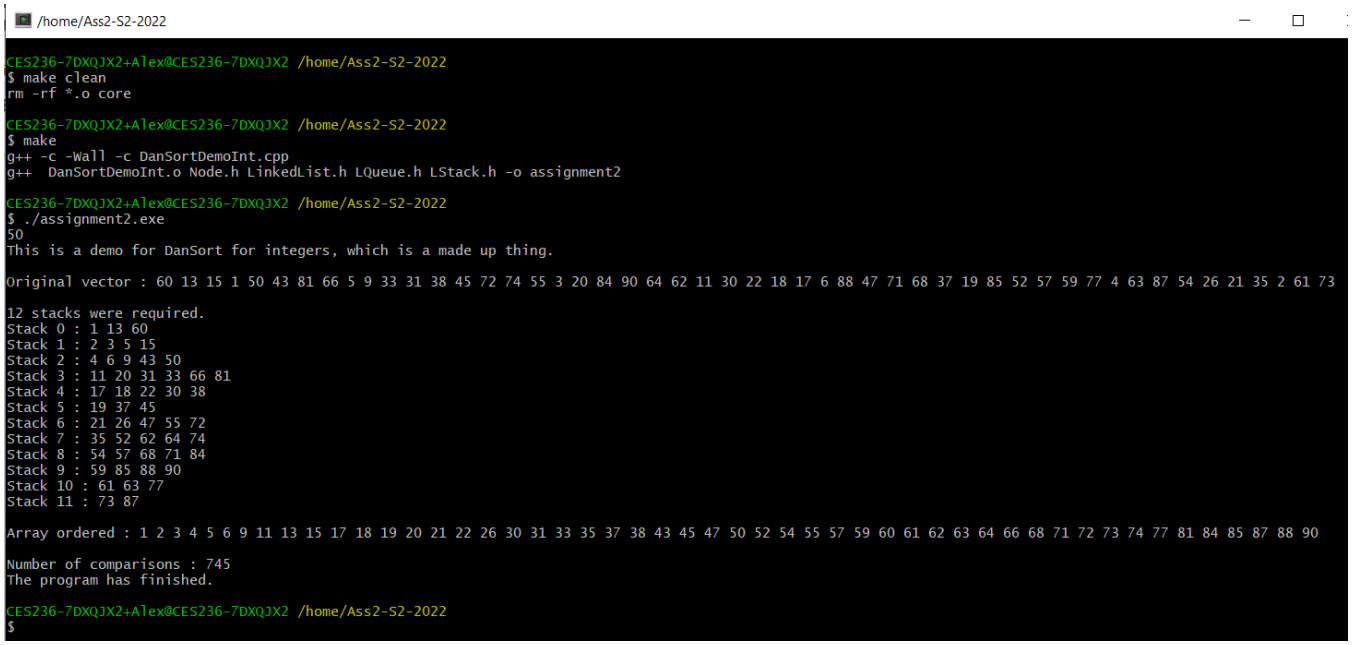
This assignment is worth 10.0 marks of your final result for the course.

Dan and Alex
2022-09-06
v.2.0.1

Compiling and running your files together with the **demo file for strings** should output the following results:


```
CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$ make clean
rm -rf *.o core

CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$ make
g++ -c -Wall -c DanSortDemoString.cpp
g++  DanSortDemoString.o Node.h LinkedList.h LQueue.h LStack.h -o assignment2

CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$ ./assignment2.exe
23
This is a demo for DanSort for strings, which is a made up thing.

Original vector : Alex Dan Steve Joel Mary Karen John Fred Michelle Eve Tom Brenda Peta
Steve Daniel Cameron Josephus Ysobel Tim Ben Anna Allan Ben

8 stacks were required.
Stack 0 : Alex
Stack 1 : Allan Anna Ben Brenda Dan
Stack 2 : Ben Cameron Daniel Eve Fred Joel Steve
Stack 3 : John Karen Mary
Stack 4 : Josephus Michelle
Stack 5 : Peta Tom
Stack 6 : Steve
Stack 7 : Tim Ysobel

Array ordered : Alex Allan Anna Ben Ben Brenda Cameron Dan Daniel Eve Fred Joel John
Josephus Karen Mary Michelle Peta Steve Steve Tim Tom Ysobel

Number of comparisons : 215
The program has finished.

CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$ ^C

CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$
```

… and compiling and running your files together with the **demo file for integers** should output the following results:




```
CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$ make clean
rm -rf *.o core

CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$ make
g++ -c -Wall -c DanSortDemoInt.cpp
g++  DanSortDemoInt.o Node.h LinkedList.h LQueue.h LStack.h -o assignment2

CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$ ./assignment2.exe
50
This is a demo for DanSort for integers, which is a made up thing.

Original vector : 60 13 15 1 50 43 81 66 5 9 33 31 38 45 72 74 55 3 20 84 90 64 62 11 30 22
18 17 6 88 47 71 68 37 19 85 52 57 59 77 4 63 87 54 26 21 35 2 61 73

12 stacks were required.
Stack 0 : 1 13 60
Stack 1 : 2 3 5 15
Stack 2 : 4 6 9 43 50
Stack 3 : 11 20 31 33 66 81
Stack 4 : 17 18 22 30 38
Stack 5 : 19 37 45
Stack 6 : 21 26 47 55 72
Stack 7 : 35 52 62 64 74
Stack 8 : 54 57 68 71 84
Stack 9 : 59 85 88 90
Stack 10 : 61 63 77
Stack 11 : 73 87

Array ordered : 1 2 3 4 5 6 9 11 13 15 17 18 19 20 21 22 26 30 31 33 35 37 38 43 45 47 50 52
54 55 57 59 60 61 62 63 64 66 68 71 72 73 74 77 81 84 85 87 88 90

Number of comparisons : 745
The program has finished.

CES236-7DXQJX2+Alex@CES236-7DXQJX2 /home/Ass2-S2-2022
$
```