ITP 365: Managing Data in C++

Homework 6: Doubly Linked List and Movie Player Due: 11/17/2017 @ 11:59PM

Goal

In this homework, you will implement a doubly linked list collection. Once you have it working, you will then simulate an old-fashion reel-to-reel movie player.

Lab Setup

- In this homework, there are quite a few files already provided to you. The majority of these files should not be edited.
- Here is a brief description of the files ***you should not edit***:
    - main.cpp – This creates a main menu, and then invokes test cases (if you select option 1) or runs the movie player (if you select option 2).
    - Tests.hpp – This defines the unit tests that will verify that your doubly linked list work. You may need to look at this file in the event that a test case fails, as it may provide insight as to what edge case you are missing. It's a GoodIdea© to examine this file carefully as understanding why your code fails a test case is important.
    - leaker.h/cpp – You should not even open these files. It is a memory leak detection library that I got from the Internet.
    - MiniCppUnit.hpp/cpp – You should not open these files, either. It implements a unit test library (that once again, I got from the Internet).
- For this homework, you ***will*** edit the following three files:
    - **ITPDoubleList.hpp** – Implements a doubly linked list with an iterator
    - **MoviePlayer.h/cpp** – This will implement the movie player behavior. You will implement most of the functionality for these classes.
- In the above three files, make sure you update the file header comments so that your name and email is listed.
- **It is extremely important you do not add any additional #include statements to any of the files. It is unnecessary to add any other includes in this assignment, and it will most likely break your project in crazy ways, because of how the leaker.h library works.**

Part 1: ITPDoubleList class

- In this part, you will implement a templated doubly linked list in **ITPDoubleList.hpp**. For each of these functions.
- **Note that although the book (and many other online references) list implementations for a doubly linked list, do not just directly copy such code, or there will be a severe penalty up to and including being reported to SJACS for academic dishonesty.**
- With the starting project, if you select option one in the main menu, you will notice that the test summary will state that it executed thirteen tests and passes zero test. Once you finish all of the functions in this section, your program should pass 14/14 tests.
- Beyond the standard doubly linked list functionality, your list should also implement an iterator to the list to keep track of a position in the list. There are functions to for the iterator as well as removing and adding at an iterator's position.
- The following is already provided for you, and ***should not be changed***:
    - Node struct
    - Member variables inside of `ITPDoubleList` (`mSize`, `mHead`, `mTail`) and `Iterator` (`mCurr`)
    - Default Constructor
    - Output Stream Operator (`<<`)
    - `toString` and `toReverseString`
- Each of the functions below need to be implemented by you. Each function has the pseudocode outline of its logic covered in lecture. You should follow this very closely to ensure your functions are correct. It is also recommended you implement them in this order:
    1. `size` – Returns the size of the list (1/14 pass)
    2. `push_front` – Insert to the front of the list (2/14 pass)
    3. `front` – Get the value at the front of the list (3/14 pass)
    4. `pop_front` – Remove the value at the front of the list (4/14 pass)
    5. `push_back` – Insert to the back of the list (5/14 pass)
    6. `back` – Get the value at the back of the list (6/14 pass)
    7. `pop_back` – Remove the value at the back of the list (7/14 pass)
    8. `clear` – Removes everything from the list (8/14 pass)
    9. Copy Constructor – Constructs the list as a copy (9/14 pass)
    10. Assignment Operator – Equals operation (10/14 pass)
    11. Iterator Basics – Creating (including `begin` and `end`) and dereferencing iterators (11/14 pass)
    12. Iterator Traversal – Moving forwards/backwards with an iterator (12/14 pass)
    13. `insert` – Using an iterator to insert into a list (13/14 pass)
    14. `erase` – Using an iterator to remove from a list (14/14 pass)
- While working on this part, you may find it easier to only run a subset of the tests. To do so you may *temporarily* comment out some of the code in **Tests.hpp**. Any number of lines 14 through 27 can be commented out to prevent a test from running to help you pinpoint an error.
- Provided you followed the order above, each function should pass an additional test. If the test does not pass, read the error message for that test, and take a look at the line in the test file. For example, if you get an error message like…

```
Test failed: Testing default constructor and size
/Users/nag/Desktop/ITP365/homework/HW6Start/Tests.hpp, line: 35
Expected:  0 But was:  2147483647
```

- …you should go to line 35 in Tests.hpp, and see what exactly it is testing. (In this particular case, it expected that the size would be 0, but it was a very large number instead). These tests will help you pinpoint exactly what's wrong with your code.
- Furthermore, before you implement the destructor, you will get an error message like:

  ```
  LEAKER: MEMORY LEAKS: 95 leaking 1520 bytes!
  ```

  Once you implement all the functions, you should no longer get the memory leak error message. If you do, make sure your deletes appear in the right spots.
- A successfully implemented Part 1 should look like the following (user input in **red**). Note how there are NO memory leak error messages.

```
Select an option...
1. Test ITPDoubleList
2. Play MOVIE
>1

+ Testing ITPDoubleList...
  - Testing default constructor and size
  - Testing insert_front
  - Testing get_front (and insert_front)
  - Testing remove_front (and insert_front)
  - Testing insert_back
  - Testing get_back (and insert_back)
  - Testing remove_back (and insert_back)
  - Tests the clear function (and insert_front)
  - Test the copy constructor (and insert_back)
  - Test the assignment operator (and insert_back)
  - Tests creating and dereferencing iterator
  - Tests iterator incrementing / decrementing (pre and post)
  - Tests insert at the iterator
  - Tests remove at the iterator
..............
Summary:
            Executed Tests:         14
            Passed Tests:           14
```

Part 2: MoviePlayer

- Now you'll implement a movie player. The movie player simulates an old fashioned reel-to-reel film player that can go backwards, forwards, and splice scenes from the film.
- To generate animation a series of frames will be displayed on the screen. Each frame should ideally appear on the screen for about 40 to 60ms (this number can be changed by user input). We'll call this the "display time" to indicate how long each frame will be displayed.
- Movies come in the form of plain text files. Each file is made up a series of frames. Here's a breakdown of the format for each frame:
    1. Each frame begins with a "separator" line. This line has only a number in it. This indicates how many long the frame should be shown. For example, if a separator has the number "7" then the following frame should appear for 7 x "display time"ms. This separator line should not be displayed to the screen.
    2. The following 13 lines (or FRAME_SIZE) in the text file make up the image (or frame) to display.
    3. The next line is the separator line for the following frame
    4. Be sure to examine the film files (they're the text files that end with the suffix ".ani"). Any text editor should be able to open them. Be sure not to change them!
- If you open up MoviePlayer.h/cpp you'll see some code already provided to you.
- There are 5 member variables:
    1. A `const unsigned` that stores the number of lines in a frame. You can use this to iterate through the movie files.
    2. An `ITPDoubleList` of `strings` that are each a frame of 13 lines of text. I'll often call this "the tape" in this document.
    3. An `Iterator` for the above structure to track where you are in the tape.
    4. A that `string` has the name of the text file that stores the movie file.
    5. An `unsigned` that holds the current frame number, or the position the viewer is seeing in the movie. Be sure to keep track of this variable as you manipulate the tape!
- You must implement all 13 member functions for the MoviePlayer class:
    1. MoviePlayer constructor: The parameterized constructor should set the appropriate member variables, ensure that the tape is empty, then call the member function `loadTape` on the inputted file name.
    2. MoviePlayer destructor: While not necessary, we'll protectively ensure that the tape is empty here.
    3. The `rewind` function: Here you'll start the tape over again, being sure to keep track of the `currFrameNum` counter.
    4. The `loadTape` function: This function should open the inputted text file. Assume the user only enters one of the ".ani" files. This function parses out the text file and makes up the frames and fills the `ITPDoubleList` with the animations.
    5. The `goFwrd` function: This function should move the tape forward 1 frame. It should also increment the `currFrameNum` counter.
    6. The `goBack` function: This function should move the tape backward 1 frame. It should also decrement the `currFrameNum` counter.
    7. The `getCurrFrameNum` function: Here you'll return the current `currFrameNum` counter.

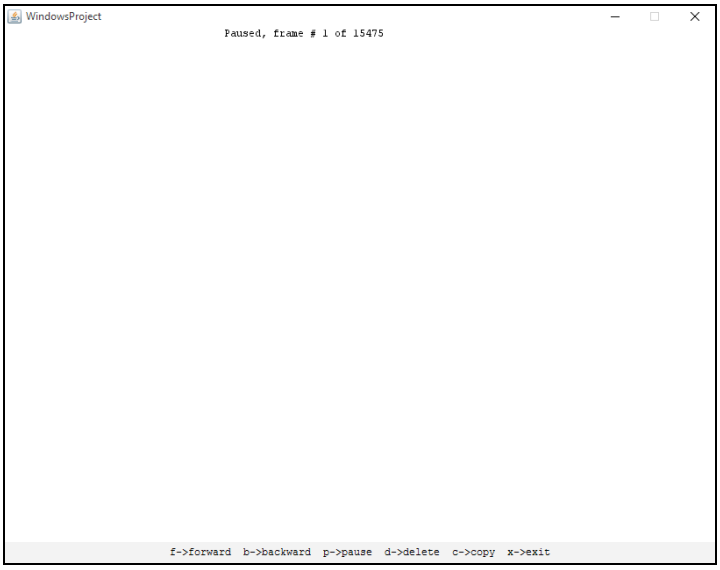
8. The `delCurrFrame` function: This function will delete the current frame and move the tape forward 1 frame.
9. The `copyCurrFrame` function: This will copy the current frame and move the tape to the newly added frame.
10. The `getNumFrames` function: Here you'll return the total number of frames in the movie.
11. The `getFrameSize` function: This function gets the number of line in a frame.
12. The two overloaded `getCurrFrame` functions:
    - One takes in no input and returns a string with the contents of the current frame.
    - Another takes in a `Vector` of `GLabel` pointers, passed in by reference. There should be 13 items in the `Vector` which is the same as the number of lines in each frame. Each `GLabel` must have its label set to the corresponding line of text in the frame.

- When asked to play a movie, main will prompt the user for a text file name and delay time. It will then create the user interface, create the MoviePlayer, and start the movie at the 1st frame. The user interface can play the movie forward and backwards. It can also pause the movie, copy the current frame and remove the current frame. The entire solution uses the Stanford Library for user interface and graphics.
- While moving through a tape, be sure to check that you're not trying to move past the beginning or end of the tape!
- **Hint:** There are shorter input files (scene1.ani, scene2.ani, and scene3.ani). Try these while testing your program. You should test to make sure you're able to successfully reach the end of the tape and beginning of the tape with no errors.
- **Hint:** Some input files have problematic input. In particular, a line may end with a backslash "\". This can be a problem for the Stanford library. You can try getting around by adding a white-space character to the end of the string.
- **Hint:** Some input files have a few empty frames. These may just be empty lines. Be sure your code detects empty lines appropriately in the `loadTape` function.
- A successfully implemented Part 2 should look like the following (user input in red). Since the movie involves over a minute of film (at 60ms delay) here's only a few screens of output (user input in red).

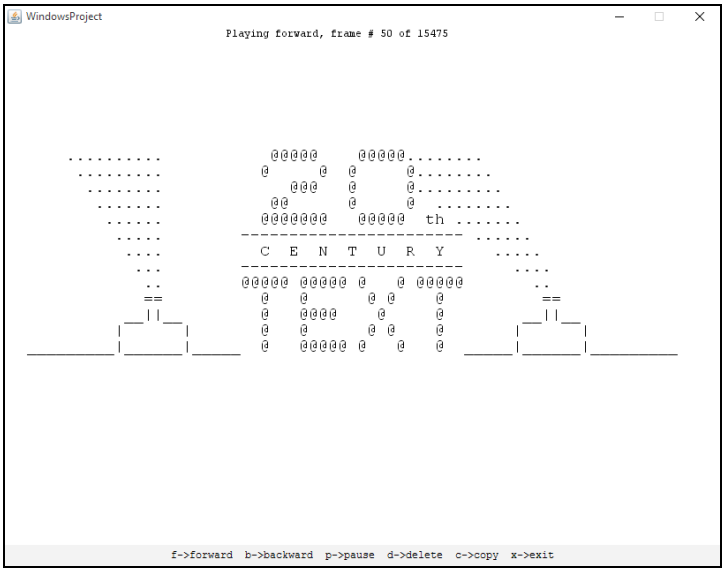
```
Select an option...
1. Test ITPDoubleList
2. Play MOVIE
>2
Input file: starwar.ani
MS delay between frames: 60
```

(continued)

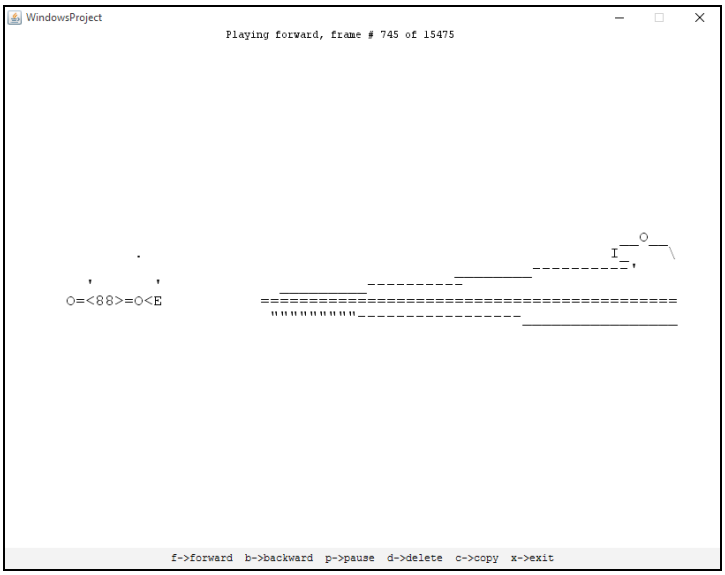
- Movie screen before movie plays:



- Movie playing:



And…



Submission

1. You must submit a ZIP file to Blackboard called **HW6Submit.zip**, using the designated submission link in the "Assignments" section. You should include in the ZIP only the files you've edited and there should be no subfolders in the ZIP.
2. Make sure to double-check that your zip file was created properly prior to submission.

Grading

Item	Points
Part 1: ITPDoubleList	
ITPDoubleList::size	1
ITPDoubleList::insert_front/back	10
ITPDoubleList::get_front/back	4
ITPDoubleList::remove_front/back	10
ITPDoubleList::clear	2
ITPDoubleList copy constructor/assignment	10
ITPDoubleList Iterator	10
ITPDoubleList::begin/end	3
ITPDoubleList::insert_after	10
ITPDoubleList::erase_after	10
No memory leaks	3
Part 2: MoviePlayer	
Properly loading the animation files	9
Correctly implementing forward/back functionality	3
Correctly implementing copy/delete functionality	2
Correctly displaying the frames (in GLabels and string)	4
Counting frames correctly	4
Comments, style, and correct use of separate files	5
Total	**100**