Lab 2: Palindromes

Valentino Lei'a
CS_124
Fall 2020

Purpose:

- Similar to our previous lab, the purpose of this exercise is to provide the students with an opportunity to implement the new concepts we've learned during this course up until now, along with our existing knowledge base of C++ concepts. Those being:
    - Stacks
    - Queues
    - Thinking about data structure (how to store the data, ie. arrays or linked lists etc.)
    - User interaction (cin, cout, data validation, user friendly messaging, etc)
    - Reading files into memory
    - Writing memory into files
    - OOP concepts (encapsulation specifically)
    - Fundamental C++ (function prototypes, variables, loops, etc.)
- Additionally we're provided another opportunity to present the finished product to a customer (professor) to simulate real world scenarios for when students get to begin their careers.

Plan & Organization:

- **What input does the program use? Why?**
    - The program is going to take a text file as input during the live demo because it's a requirement given by the professor
    - The end user's name is going to be taken as input from the console because it's a good user experience to ask the user their name and say hello for this program
    - The user will have the option to select between choosing a file or typing in their own palindrome, both of these options are input for the program and are requirements
    - A string provided by the user is going to be input that meets the requirement of this assignment
    - Strings from a .txt file is going to be input that is stored in memory as specified by the requirements
    - The response of the user wanting to repeat testing palindromes is input that is required for this program
    - BONUS: the user can upload their own text file
    - To touch a bit more on the why of all of these inputs, it's just a good user experience to be able to meet the need or common interactions that they will have with the program. You never want an end user to feel like they can't execute something with the product they're using and feel unsatisfied with the program.
- **What variables will you need? Why?**
    - Definitely going to need the string class to use strings when storing the ***strings*** from the file and user input
        - A little simpler than using cstring :)

    - I plan on using linked lists as a data structure to store palindromes in an ordered list
        - This will make it pretty easy to implement stack and queue for this assignment seeing as we got practice with it in a previous assignment :)
    - Plan on using some integers when I use for loops, or as a counter when I use a while loop, depending on the situation.
    - Definitely going to need an object of the fstream class to read and write files
    -
- **What constants, if any, will you need? Why?**
    - Constants, hard maybe for now depending on how I plan to clean data and store file names that already exist in the file path before runtime
- **What functions do you use? How did you decide what operations go into each function?**
    - A class that mainly handles user interaction
        - Functions for getting user input and printing to the console
        - I plan to have a function that validates user input with error handling
            - Overloaded to handle different data types
                - I've only used templates one time before so I still use this method (pun intended)
        - Setters and getters if needed
        - Constructors for each class
    - A class for data manipulation
        - A constructor that creates a linked list
        - Function that modifies that list (add in order, end, begin)
        - Function to prints the list
        - Deconstructor to free the allocated memory
        - I'm going to have another function that removes unwanted characters from a string
        - One function to read data from a file into memory
        - Similarly, a function to write data from memory into a file

```cpp
/*



*/

#include <iostream>
using namespace std;

// function prototypes

class files{

    private:
    public:
};

class fileInputAndOutput: public files{

    private:
    public:
};

class userInterface{

    private:
    public:
        userInterface();
};

class dataValidation: public fileInputAndOutput, public userInterface{

    private:
    public:
};

class linkedListFunctions{

    private:
    public:
        linkedListFunctions();
        ~linkedListFunctions();
};


struct listNode{

    listNode *next;
};




int main(){

    cout << "asdfdsasd world\n";

    string name;
    cin >> name;
    cout << name << endl;

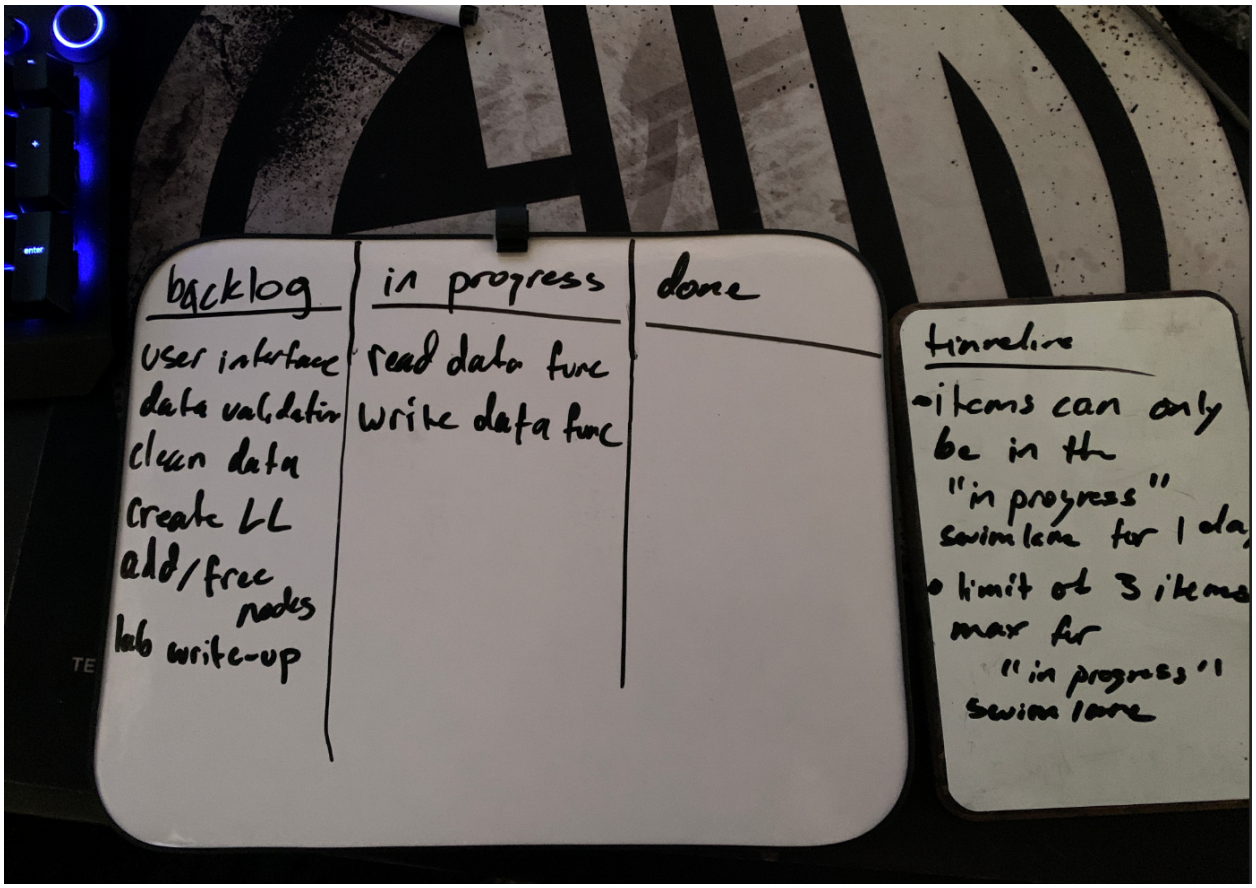
}
```

- Is how I initially started out before writing this report
    - And no I don't use namespace std in the final product. That was just there for the sake of starting this piece out and being lazy not wanting to type out std:: more than 1x

- **<u>How do you decide that your program is complete?</u>**
    - I decide that the program is complete by first meeting every requirement listed in the requirements document.
    - If every criteria has been satisfied, then I can move onto nice to have functionality if time allows. Those things being the following:
        - Is there any undesired or odd functionality that exists

        - Does it need fixing or to be addressed?
    - Was there anything I “cheap’d” out on in terms of bad coding practice but just went ahead for time management
        - Can I go back and fix it to make it more efficient (cheaper or faster)

- **Timeline:**
    - Below is a representation of small tasks that I broke out for this assignment that I’d like to get done. I put them in a backlog swimlane and move them over to in progress and done respectively.
    - The timeline or guideline I have for this is:
        - Items can only be in the in progress swim lane for 1 day max
        - Limit of 3 items maximum for the in progress swim lane



Development Process:

- My development process was fairly simple. I took the plan I had created from part 1 of this lab, and built all of those classes. Below describes each class and the attributes/functions for the class. It was pretty easy to do this considering most of the work had already been done in assignments leading up to this lab, or as part of the previous lab. So I really copy/pasted a lot of this code from previous work done in this course. For the remaining that wasn’t copy/pasted, it was pretty straightforward to write. I

pretty much did one class at a time until I was finished with the program. I then made sure that each requirement was met.

- **Build the user interface (interaction and processing):**
    - My design for this class was for any code that interacts with the user should live here. I used inheritance in this class because it made sense to be able to call the dataValidation functions when taking user input here.
    - ```
      #include "dataValidation.h"
      #include "linkedList.h"
      #include "fileInputAndOutput.h"

          class userInterface: public dataValidation{//, public fileInputAndOutput{

          private:

              int menuOption, textFileOption;

              std::string name;
              std::string userInputPalindrome;

          public:

              userInterface();

              void setUserName();
              std::string getUserName();
              void greetTheUser();

              void createMenu();
              void setUserInputPalindrome();
              void createTextFileMenu();
              void createOptionalTextFileMenu(fileInputAndOutput &);

              void createStackFromUserInputPalindrome(linkedList &);
              void createQueueFromUserInputPalindrome(linkedList &);

              void repeat();

          };

      #endif
      ```

- **Build the processing for manipulating text files:**
    - This class handles reading data to memory from a file, writing data from memory to a file, set the name of a file from user input, get the name of the file input by the user, get hard coded name of file1, get hard coded name of file2. I used inheritance in this class because I wanted this class to inherit the functions,
      ```

constructors, and attributes of those classes. I felt it would make the flow of the program efficient and easy to use.

```
#include "linkedList.h"
#include "dataValidation.h"

using std::string;

    class fileInputAndOutput: public linkedList, public dataValidation{

        private:

            const string samplePalindromes = "PalindromeSamples.txt";
            const string tinosSamplePalindromes = "PalindromeTest2.txt";
            string userInputFileName;

        public:

            void setUserInputFileName(string);
            string getUserInputFileName();

            const string getPalindromeSample();
            const string getTinosCoolPalindrome();

            void writeToFile(fileInputAndOutput &, string);
            bool readFromFile(string);

    };

#endif
```

- **Build the validations for the programt:**
    - The purpose of this class was to validate all user input, check strings for palindromes, remove unwanted characters from strings, and create a stack/queue

```
#include "linkedList.h"

using std::string;

    class dataValidation{

        private:

        public:

            string removeUnwantedCharacters(string);
            void validateUserInput(int &);
            bool checkPalindrome(linkedList &, linkedList &);

            void createStackFromUserInputPalindrome(linkedList &, string);
            void createQueueFromUserInputPalindrome(linkedList &, string);

            string validateStrings(string);
            bool validateYesOrNo(char);
    };

#endif
```

- **Build the LinkedList class:**
    - The purpose of this class was solely for functions relating to linked list. Adding nodes at the beginning or end of the list, popping nodes from the beginning, freeing the list, creating the list, returning the head, printing the list.

```
#include "ListNode.h"

    class linkedList{

        private:

            ListNode *head;

        public:

            linkedList();
            ~linkedList();

            ListNode* getHead();

            void addNodeAtBeginning(std::string);
            void addNodeAtBeginning(char);
            void addNodeInAscendinglOrder(std::string);
            void appendNode(std::string);
            void appendNode(char);
            void removeNodeAtBeginning();
            void printList();
    };
#endif
```

- **Build the struct for each node:**
    - Each node needs to have a string to store a palindrome and a pointer to the next node.

```
    struct ListNode{

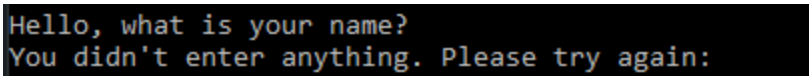
        std::string data;

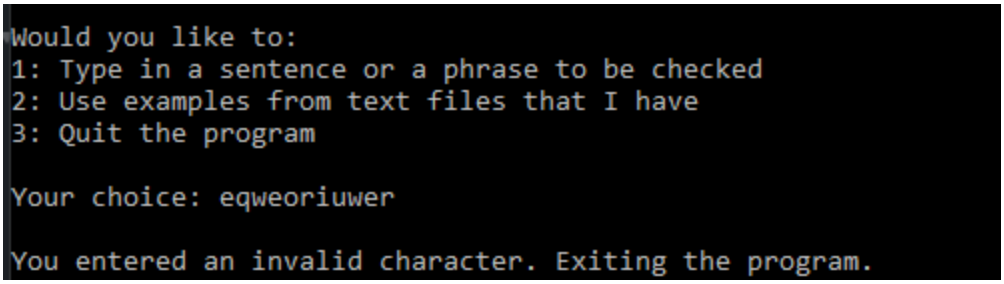
        struct ListNode *next;

    };

#endif
```

Product:

- Program execution:
    - The user will be asked to enter:
        - their name
        - a palindrome
        - menu option to repeat the program
        - menu option to select a text file
        - menu option to enter a palindrome or test a file
        - a name of a file
    - The following is printed to the console:
        - Ask the user for their name
        - Greet the user
        - Program description
        - Menu to select file or input a palindrome
        - Menu to select 1 of 2 files
        - List of palindromes parsed from file
        - User input palindrome (if it is or isn't indeed a palindrome)
        - When the program is going to be exited
        - Farewell to the user
    - What is processed during the program
        - User input is validated
        - File contents stored in memory
        - Memory written into a file
        - If a string is a palindrome
        - If the user wants to repeat the program
        - Creating a stack and queue with palindromes
        - Popping stack and queue
        - Printing linked lists
- Test cases: (this part wasn't hard for me seeing as I work as a QA Engineer…)
    - During user input
        - What data can you enter
        - Useful messaging when validation user input
            - 

            - 

```
Your choice: -1
Invalid entry, try again: 4
Invalid entry, try again: 55

You failed to enter a valid option 3 times in a row. Exiting the program.
```

```
Would you like to do another? (y/n)
we
You entered an invalid option. Exiting the program.
```

- When printing out menus
  - Again context of where the user was
  - User friendly messages
  - Good line spacing as to not clutter the console

```
Would you like to:
1: Type in a sentence or a phrase to be checked
2: Use examples from text files that I have
3: Quit the program

Your choice: _
```

```
Would you like to output the palindromes to a seperate text file? (y/n)
y
Enter a file name: testfile.txt

Would you like to do another? (y/n)
n

Peace out dude!
```

Pitfalls:

- I didn't really run into any pitfalls with this lab. I felt it was very easy to understand, plan and execute. I *could* say that thinking through how I wanted to verify a palindrome was a challenge, but I wouldn't classify it as a pitfall. It really just required me to think through a way to compare two strings in the context of my program.

Possible Improvements:

- I honestly wouldn't change anything about this assignment. I felt my planning and execution went very well. Overall it only took me about 5-6 hours to complete from start to end. One thing that can be improved, kind of falls outside of this assignment, is my work life balance. I found myself developing this lab from like 10pm onwards two nights in a row. Better setting aside time for studying and school after work and time for exercise or leisure would have me working on a healthier schedule for my well being. Besides that I wouldn't change anything about how this lab went.