CS2201
Fall 2018
Assignment No. 3

Purpose: Gain experience in using and manipulating linked lists.

Background on DNA, Restriction Enzymes, and PCR:
This assignment builds upon ou nting a DNA strand by using a sequence of characters. In this lab, we will enhance our representa use of arrays and replace them with a linked list. This will allow our DNA strand object to grow and o longer will we need to shift data down/up in the array when we cleave/splice, rather we will be gments of characters with just a few pointer manipulations.

The Assignment:
Just as in the prior assignment, nt a DNA strand, but now rather than using a partially-filled array of characters, we will use a linked list. This will allow us to represent very large DNA strands and also allow us to easily increase/decrease the size of the strand during execution.

The functional specification below will lead you through the steps necessary to convert your DNA_Strand class to using linked lists. We will not be adding any additional functionality to the class, just changing the underlying representation.

The DNA_Strand.h file describes all the functions to be implemented, as well as the structure of a **DnaNode** used for our linked list.

Functional Specifications:
You will be supplied the class declaration file: **DNA_Strand.h.** The file contains the declaration of a set of functions to manipulate linked lists representing DNA. You will also be provided with a **DNA_Strand.cpp** file, where most of the methods have empty bodies or simply return some default value. You are free to alter the private data in the class, but you are not allowed to change anything in the public section of the class. Your task will be to implement all the specified functions in the file **DNA_Strand.cpp**.

The **DNA_Strand.h** file is identical to that of project #2 with the following exceptions:

1. We define a **DnaNode** structure

```
// the structure of a linked list node contains a single char and a next pointer
struct DnaNode
{
   char val;
   DnaNode *next;
};
typedef DnaNode* DnaNodePtr;
```

   We have defined this struct outside the DNA_Strand class so that it is visible to users of the class. This was necessary since several new methods take DnaNodePtr parameters or return DnaNodePtr values. You are not allowed to change the structure of this node.

2. We eliminated the DNA_Strand constructor that takes a size parameter. It is no longer needed.

3. We have added an additional search() method, which will be useful in this project:

```
// search with start position specified by a DnaNodePtr
// Look for target in current DNA strand and return a pointer to it
// Return nullptr if not found.
DnaNodePtr search(DnaNodePtr startPtr, const std::string & target) const;
```

4. We have added additional cleave() and splice() methods, which will be useful in this project (the other cleave & splice methods should be written in terms of these two methods). Also the cleave() and splice() methods that take a starting position specified by an index are now void functions, rather than returning an index.

```
// cleave with start position specified
// Start position is specified with a DnaNodePtr.
// If no cleave is performed, returns nullptr
// Otherwise, return pointer to next node after the cleave, ie,
//   returned ptr points to the cleaved sequence (could be
//   nullptr if the cleave is at the end of the strand).
DnaNodePtr cleave(DnaNodePtr start, const std::string & target);

// splice (also accepting position specified with a DnaNodePtr)
// If no splice is performed, returns nullptr
// Otherwise, return a pointer to next node after the inserted sequence
DnaNodePtr splice(DnaNodePtr start, std::string target, std::string insertSequence);
```

5. We eliminated the grow() method, as it is no longer needed since we can insert nodes into the linked list at any point.

6. Our private data no longer contains a maxDNA component (since we no longer need to keep track of an array size). We still maintain a mySize component, which saves us from having to walk the list to determine how many characters we are storing. And myDNA is no longer a char* pointer, but is now a DnaNodePtr, and it acts as the head pointer of the linked list.

Using CLion for this lab: You are to download a zip file that is provided with this specification. The zip file contains starter code for this assignment in the form of a CLion project. You ***must*** unzip/extract the files before you work on them. After you unzip the file, start up CLion and select *Open Project*. Navigate to & select the folder created by unzipping the file, and open it in CLion. When it is done loading all symbols (watch the progress bar at the bottom) you should be able to compile it. The code as provided compiles (though it contains a bunch of temporary code to make the compile happy) and the program should run as given – though the results are incorrect since required functionality is missing. Note that your final code is expected to compile cleanly – no warnings or errors.

You may go about developing this project in any manner you see fit. Here is the strategy I used which I found helpful:

1. Begin working on one or two methods at a time. Start with your constructors and Big Three. There will be times when it will be useful to review how you wrote that same method in project #1 or project #2. You will need to modify the code from the earlier projects to use linked lists rather than arrays. In some cases, the modifications are fairly straightforward. In other cases, the methods have to be completely rewritten from scratch. Be careful: do not constrain your thinking to processing the linked list like arrays – linked lists give us much more power to do things efficiently. In particular, do not continue to think in terms of indices into the DNA strand (since we can no longer do direct indexing), but instead think in terms of pointers to nodes whenever you can.
2. Once these methods compile cleanly, test them thoroughly but adding new test code to the driver or copying test code from your project #2 test driver. Once they operate correctly, go back to the previous step and start on the next method. Lather, rinse, repeat until all methods are working.

Other details:

Here are a few notes that might be helpful:

1. Continue to use type `size_t` when appropriate.
2. You are required to use the base initialization list for your constructors as much as possible.
3. For methods that use an index into the strand, we will continue to use zero-based indexing; i.e., the first character of the strand is at index zero.
4. Since some of our overloaded methods can now take either an index or a pointer as a parameter, the compiler will flag certain calls to these methods as ambiguous. For example, the search method is overloaded to accept a starting position specified as either an index (type size_t) or a pointer (type DnaNodePtr). If you call the function with the literal value zero the compiler cannot tell if you are passing a zero index or a NULL pointer. This is a result of the fact that a NULL pointer is the value zero (this is not an issue for the value `nullptr` since it has an explicit pointer type). To resolve this, you need to type cast the literal value zero in the call; e.g., use search((size_t)0, "ATG") to search with a start index or search((DnaNodePtr)0, "ATG") to start with a NULL pointer [for us, we will always replace the

second option with search(nullptr, "ATG")]. This ambiguity does not occur when you are using variables, since the variables have a declared type.

5. You are free to add helper methods to the private section of your class, though I did not find the need for any.
6. As with projects #1 & #2, you are not allowed to convert your DNA_Strand to a string object so that you can search or edit the string. Rather you are to implement the methods in this assignment by operating directly on the linked list.
7. Additionally, you will not receive full credit if you treat the linked list as an array (by constantly calling set() & get() with indices). This is wasteful, as each call to set() & get() must perform its own walk of the linked list. The purpose of the assignment is to learn how to directly process linked lists.
8. In lecture, we stored the dig in reverse order (so the ones digits was at the head of the list). You do **not** want to store your D n reverse order, but they need to be stored with the first character at the head of the list.
9. Be sure to review the gradi bmission so that you do not make the same mistakes in project #3.

Design: Recall from our class d e several design decisions to be made. Your decisions may have a big impact on how easy/hard it d/or debug certain functions. Consider your design alternatives *before* you begin coding.

Final write-up: When you have completed the assignment, create a **README** document (a .txt text file or a .doc Word document), and in it answer the following questions:

1. State your name and email address.
2. After reviewing this spec and the .h file, please estimate/report how many hours you think it will take you to complete this project. [This is just an estimate and does not affect your grade.]
3. How many hours did you actually spend total on this assignment? [This information will not affect your grade.]
4. Did you encounter any impediments that prevented you from making progress on this assignment?
5. What did you like or hate about this assignment?
6. Do you have any suggestions for improving this assignment?

Submission for grading:

The files you need to submit include **DNA_Strand.h**, **DNA_Strand.cpp** *and* your test driver (likely named **DNAtest.cpp**), plus your final **README** write-up document. Please submit ONLY the four files individually; do not zip the four of them together into a single .zip file, and do not zip up your entire project directory (it is too large). You can submit the files by visiting the assignment page in Brightspace (click on the assignment name), scroll down to the "Submit Files" section, and add the files by clicking on the "Add a file" button and finding the files to attach.

After submitting your homework, it is good practice to verify your submission. Revisit the assignment page in Brightspace and make sure that your files were successfully submitted. Then click on each file to open it up so that you can verify that you submitted the correct file, as opposed to some older version of the file. It is your responsibility to insure the correctness of your submission.

Grading:

This project is worth 50 points. Your grade on this project will be based on the following:

1. The correctness of your methods implemented in the DNA_Strand.cpp file.
2. The use of good programming style. No lines of code past column 100, proper indentation, proper use of curly braces, etc. See the style document posted to Brightspace under Course Resources.
3. Eliminating redundancy in your code. If you fail to utilize other methods that you have written and have repeated functionality in multiple methods, you will lose points.
4. The thoroughness of your testing performed in the DNAtest.cpp file. Note: the thoroughness of your testing often has a great impact on the correctness of your code (see item #1 above).
5. Appropriate responses to all questions in the README file.

Caution: This lab requires that you manipulate pointers more than any previous lab. Minor mistakes can cause all sorts of unpleasantries. You should plan on **TRIPLE** the debugging time. Work on only a few methods at a time, getting them fully debugged before moving on to other methods. Remember to consider edge conditions.