CSCE 240 Final Project Report

December 5, 2016

Team members:

This project was conducted as an individual assignment by:

1. Jeffrey Cocklin Computer Information Systems, jcocklin@email.sc.edu

General problem:

Assembly of small gene fragments into one contiguous gene sequence based on at least 20 base character overlaps.

An overview of the solution:

My approach to this problem is to first read in one sequence from a file containing the gene Sequences, and store it as a Gene object. The Gene object extends the String class, and provides additional functions necessary for assembling the gene sequence. The Gene object will act as the initial seed of the sequence to be constructed. The remaining sequences will be read into a queue, and each sequence in the queue will be compared to the seed to determine overlap. If the overlap is less than 20, the head node of the queue containing the sequence that was compared is sent to the back of the queue. If the overlap is 20 or above the sequence is concatenated to the seed, and the head node is dequeued. This process is repeated until the queue is emptied, whereupon the complete assembled sequence will be written to an output file.

Software design:

The final implementation of the Simple Sequence program consists of three object classes: String, Gene, DnaQueue, and one driver class main. The String class was constructed from previously assigned projects. The Gene class inherits from String, providing the methods and attributes necessary to compare sequences. The DnaQueue class was used to contain sequences, other than the seed, in a Node containing a character array of size 100. During the implementation many new changes were made to the original specifications. Execution of program now uses an Object File detailed in the README.txt. Error codes 102 and 104 are no longer included in the final implementation. The Node Class contains a char[100], in place of a gene Object, due to problems that arose from memory allocation. The function enqueue, in the DnaQueue class, now takes in const char [], instead of a gene, as a result. Gene.h now contains a new attribute int match. Several new functions have also been added to Gene.h:

```
int operator == (const char gene2[]);

friend int operator == (const char gene2[], Gene &curSeq);

Gene& operator = (const Gene &gene2);

Gene operator + (const char gene2[]);

friend Gene operator + (const char gene2[], Gene &curSeq);

friend ofstream& operator < (ofstream &fOut, const Gene &gene);

char getData( int index)const;

int getMatch()const;

void setMatch(int num);
```

In main.cpp functions changed to:

```
void readFile( ifstream &,ofstream &,DnaQueue &);
void sequence(DnaQueue &, Gene &, ofstream &);
void writeOut(ofstream &,const Gene &);
```

The various changes are now reflected in the current UML diagram featured below. To better highlight these changes the original UML diagram has also been included.

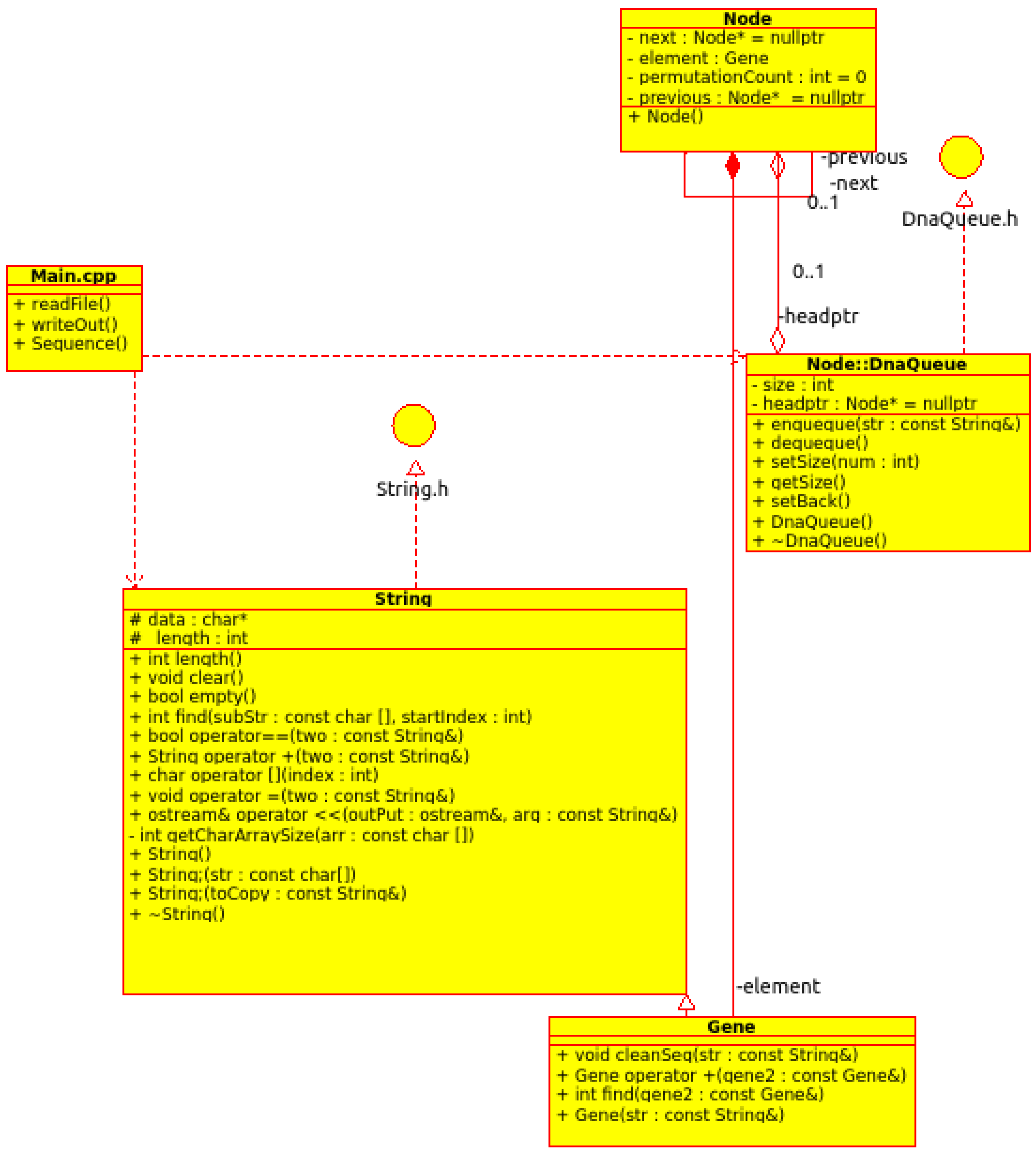

Node
- next : Node* = nullptr
- element : Gene
- permutationCount : int = 0
- previous : Node* = nullptr
+ Node()
-previous
-next
0..1
DnaQueue.h
0..1
-headptr
Main.cpp
+ readFile()
+ writeOut()
+ Sequence()
Node::DnaQueue
- size : int
- headptr : Node* = nullptr
+ enqueue(str : const String&)
+ dequeue()
+ setSize(num : int)
+ getSize()
+ setBack()
+ DnaQueue()
+ ~DnaQueue()
String.h
String
# data : char*
# length : int
+ int length()
+ void clear()
+ bool empty()
+ int find(subStr : const char [], startIndex : int)
+ bool operator==(two : const String&)
+ String operator +(two : const String&)
+ char operator [](index : int)
+ void operator =(two : const String&)
+ ostream& operator <<(outPut : ostream&, arg : const String&)
- int getCharArraySize(arr : const char [])
+ String()
+ String;(str : const char[])
+ String;(toCopy : const String&)
+ ~String()
-element
Gene
+ void cleanSeq(str : const String&)
+ Gene operator +(gene2 : const Gene&)
+ int find(gene2 : const Gene&)
+ Gene(str : const String&)

Figure 1: Original UML diagram of my proposed software.

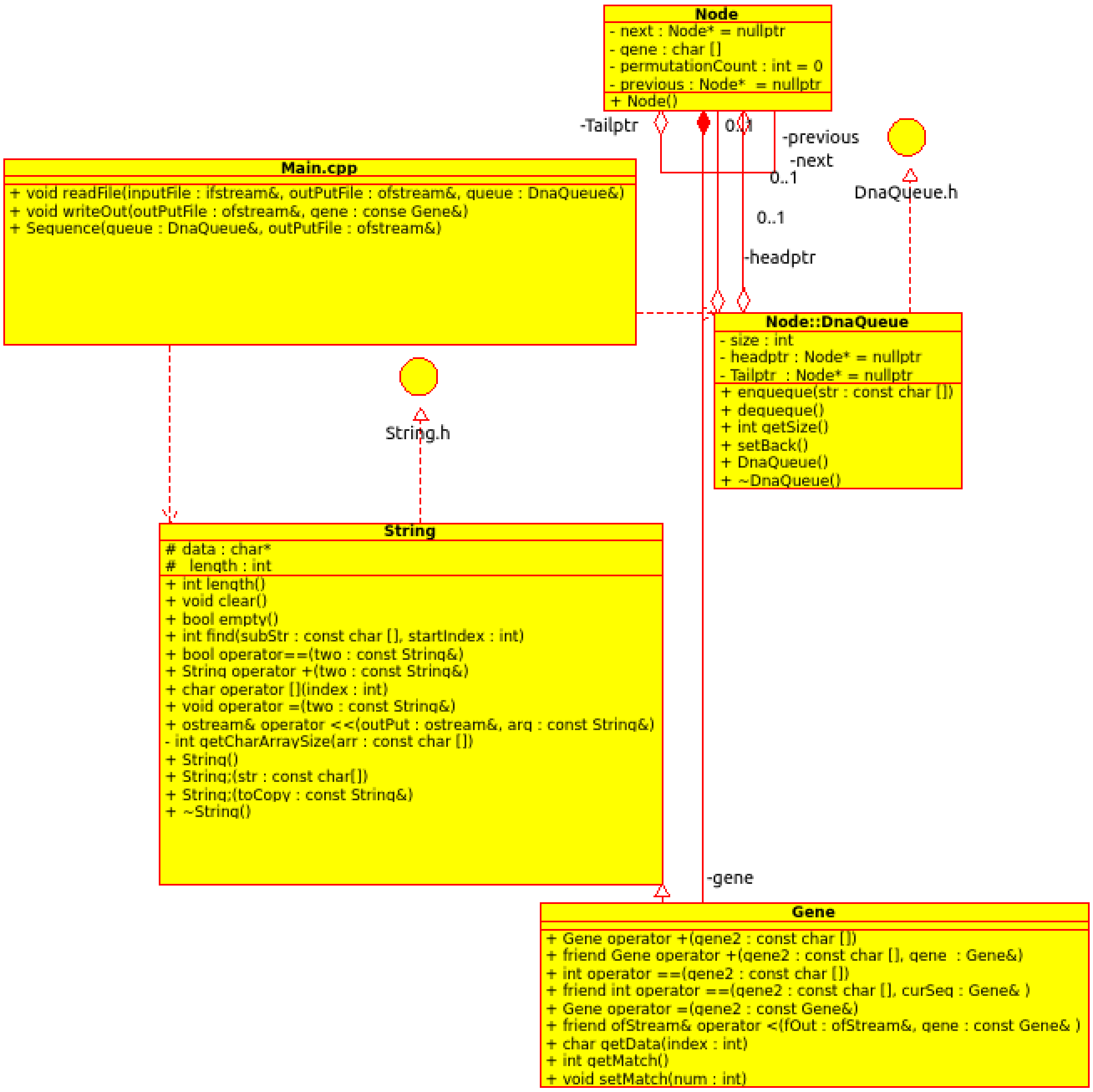

Node
- next : Node* = nullptr
- gene : char []
- permutationCount : int = 0
- previous : Node* = nullptr
+ Node()
-Tailptr
0..1
-previous
-next
0..1
0..1
-headptr
DnaQueue.h
Main.cpp
+ void readFile(inputFile : ifstream&, outPutFile : ofstream&, queue : DnaQueue&)
+ void writeOut(outPutFile : ofstream&, gene : conse Gene&)
+ Sequence(queue : DnaQueue&, outPutFile : ofstream&)
Node::DnaQueue
- size : int
- headptr : Node* = nullptr
- Tailptr : Node* = nullptr
+ enqueue(str : const char [])
+ dequeue()
+ int getSize()
+ setBack()
+ DnaQueue()
+ ~DnaQueue()
String.h
String
# data : char*
# _length : int
+ int length()
+ void clear()
+ bool empty()
+ int find(subStr : const char [], startIndex : int)
+ bool operator==(two : const String&)
+ String operator +(two : const String&)
+ char operator [](index : int)
+ void operator =(two : const String&)
+ ostream& operator <<(outPut : ostream&, arg : const String&)
- int getCharArraySize(arr : const char [])
+ String()
+ String:(str : const char[])
+ String:(toCopy : const String&)
+ ~String()
-gene
Gene
+ Gene operator +(gene2 : const char [])
+ friend Gene operator +(gene2 : const char [], gene : Gene&)
+ int operator ==(gene2 : const char [])
+ friend int operator ==(gene2 : const char [], curSeq : Gene&)
+ Gene operator =(gene2 : const Gene&)
+ friend ofStream& operator <(fOut : ofStream&, gene : const Gene&)
+ char getData(index : int)
+ int getMatch()
+ void setMatch(num : int)

Figure 2: UML diagram of my final implemented software.

<u>**Time complexity analysis:**</u>

The time complexity of my algorithm was established by using the Linux date function as a means of time stamping the start of execution and the end of execution of the algorithm. The cases observed according to input n, where n represents the number of sequences to be read in, include: n={5, 10, 100, 1000, 10000, 1000000}. The recorded time complexity is represented as the difference between end time and start time. Figure 3 and 4 depict the time complexity in graph and table form respectively.

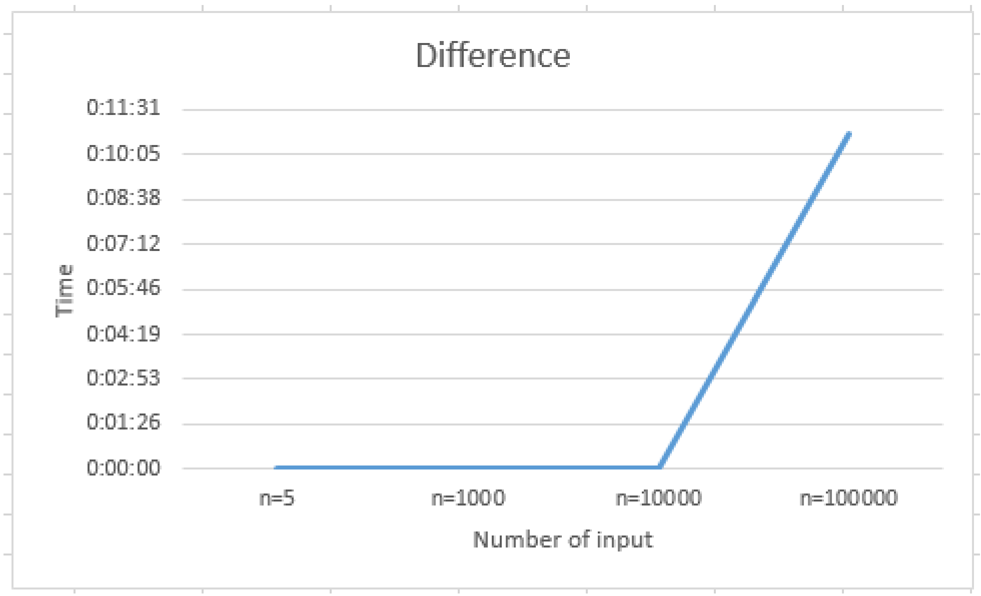
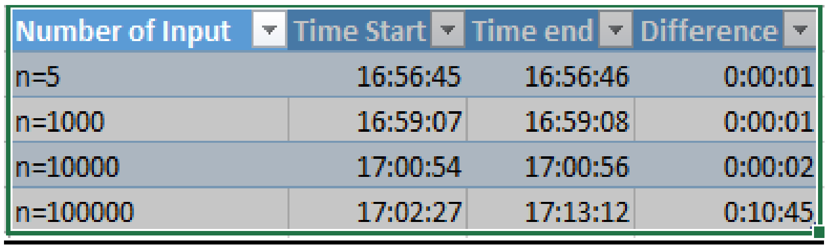


Figure 3: Execution time analysis of my approach as a function of number of sequences.



Number of Input	Time Start	Time end	Difference
n=5	16:56:45	16:56:46	0:00:01
n=1000	16:59:07	16:59:08	0:00:01
n=10000	17:00:54	17:00:56	0:00:02
n=100000	17:02:27	17:13:12	0:10:45

Figure 4: Execution time analysis of my approach as a function of number of sequences.

Peer evaluation:

Due to the fact this project was carried out alone this section is not applicable

Challenges:

The number one Challenge I faced in the creation of this program,
was tracking down memory leaks. The second challenge the project posed was, how to write the algorithm comparing the Gene object to the gene sequences. The third most challenging aspect, involved creating the data structure to house the gene sequences. The challenge came from making sure the pointers were shifted correctly, and the data pointed to was released without error.