Jeffrey Cocklin
CSCE 240 section 2
Final project Part A

# Simple Sequence Documentation.

# Table of Contents

## Problem Description:

The vast complexity of the human body can lead to a multitude of diagnoses and treatments. Physicians must try to sort through a myriad number of possible causes, when evaluating the symptoms of a patient. This often results in physicians making an educated guess about what may be the cause of a problem, and may even lead to an ineffective treatment strategy.

## Solution:

With the successful mapping of the human genome, it now possible for genetic abnormalities to be identified. Next generation sequencing holds the key to identifying and treating aliments on a patient by patient basis. By creating software that facilitates the process of genetic sequencing, it will be possible to lower the costs and time associated with treating an individual. The “Simple sequence”, software outlined in this document, is designed to accomplish that task.

## Overview:

This documentation includes problem statement, the described solution, along with an Overall description of the expected environment. This document will also include Specific requirements, and functional requirements that must be meet for the program to run. In addition a section about Error codes, as well as terms and definitions are provided. An estimation of time complexity is also provided. A UML diagram is provided, accompanied by a design description that describes the functions in the UML diagram. The Use cases will be described, followed by a test case for each of the use cases.

## Overall description:

### Product perspective

- Provides a simple program for the user to input file DNA sequencing information a constructed DNA sequence
- Limits user interaction to prevent erroneous input and output.
- Ability to read files.
- Ability to write files.
- Ability to perform DNA sequence assembly.

### User Characteristics

- The User is expected to be able to compile and run this software with the appropriate input.
- It is not required that, the User possess programming knowledge.
- The User is expected to have knowledge of basic file structure, and command-line usage.

### Constraints

- Time dedicated to testing, is the primary constraint facing this projects development, this is an unfortunate consequence of undergraduate studies.

### Assumptions

- It is assumed that the software is being run on a computer with a C++ compiler.
- The code is compiled with C++11 flag.

## Specific Requirements:

This section will specify what the program requires to run successfully.

The necessary interfaces for execution are as follows: String.h, DnaQueue.h.

The necessary classes include: String.cpp, DnaQueue.cpp, Gene.cpp

The necessary include Files: one file to be read by the program.

## Functional Requirements:

This section will detail what is required of the program to function successfully, and provide the necessary conditions to meet those requirements.

**Feature 1**

Requirement Name: Read file.

Requirement #: FR-0

Introduction: In order for system to function, it must read input from a file specified by the user.

Inputs: filename.

Requirements Description: Once the program has been compiled, it must be invoked using two arguments: ./a.out and a file name.

- If the number of arguments is greater than or less than two, the program will display an error message and quit.
- If the file cannot be read, the program display an error message and quit.

Outputs: File contents to console (For Test Case 1 only). Otherwise file contents is stored in an instance of DnaQueue.

Test Cases: 1

**Feature 2**

Requirement Name: Write file.

Requirement #: FR-1

Introduction: In order for system to function, it must read input from a file specified by the user and provide output in the form of a file.

Inputs: filename.

Requirements Description: Once the program has been compiled, it must be invoked using two arguments: ./a.out and a file name.

- If the number of arguments is greater than or less than two, the program will display an error message and quit.

- If the file cannot be read an error message will be displayed and the program will quit.
- If the file cannot be written an error message will be displayed and the program will quit.

Outputs: File contents to output file(for Test Case 2 only). Otherwise complete assembled sequence, or error messages sent to output file.

Test Cases: 2

**Feature 3**

Requirement Name: Sequence input.

Requirement #: FR-2

Introduction: In order for system to function, it must be able to concatenate multiple Sequences into a single assembled sequence.

Inputs: element, currentSeq

Requirements Description: Once the program has been compiled, it must be able to identify character overlap, and concatenate two sequences without duplicating the overlap segment.

- If the two segments don't overlap currentSeq will contain only the sequence prior to the unsuccessful comparison.

Outputs: File contents to console

Test Cases: 3

## Errors:

In this section, the possible errors that can be encountered will be described.

ID:101 File not found.

This error occurs when the program tries to read a file that, is not within the same file directory as the program, or the file does not exist.

ID: 102

Program accessed a char outside string bounds

ID: 103 Cannot open write file.

This error occurs when the file that displays output cannot be written to.

ID: 104 Segmentation fault

This error occurs when a memory location is illegally accessed.

## Terms and Definitions:

FR- Functional Requirement.

N/A- Not avialable.

User- Anyone who interacts with the software.

ID- Identification.

Feature- A function that the program carries out.

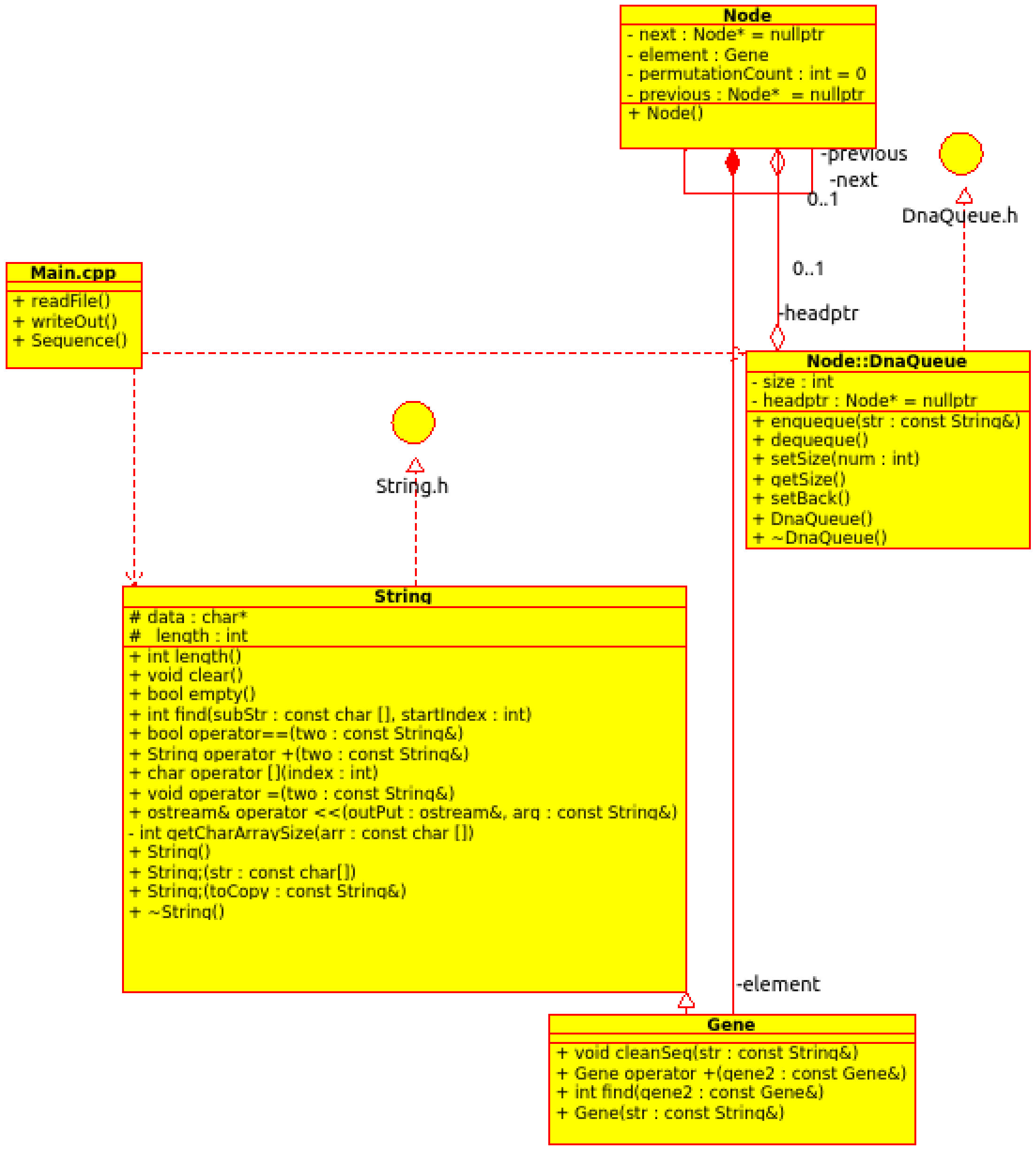
Interface- The header file that contains the prototypes of a class.

Class- The definitions of prototypes contained in a header file.

## Time Complexity:

Let N, represent the number of strings to be read into the queue. Assuming all of the sequences to be assembled are needed to construct a complete sequence. The program should take N^2N time. This prediction is solely based on the search functions nested for loop.

## UML Class diagram:


Node
- next : Node* = nullptr
- element : Gene
- permutationCount : int = 0
- previous : Node* = nullptr
+ Node()
-previous
-next
0..1
DnaQueue.h
Main.cpp
+ readFile()
+ writeOut()
+ Sequence()
0..1
-headptr
Node::DnaQueue
- size : int
- headptr : Node* = nullptr
+ enqueque(str : const String&)
+ dequeque()
+ 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&)

## Design Description:

### String class

String()- default String constructor.

String(const char str[])- creates a String from a cString argument.

String(const String& toCopy)- Copy constructor.

char * data- pointer to dynamic array

int _length- the number of chars in the String.

**i**nt Length() const- Function returns the number of characters in a String.

void Clear()- Function sets String size to zero.

bool empty()const-Function returns a true if String is empty.

int find(const String & subStr, int startIndex)const- Function returns an int representing the start index of substring, in the Calling String. Returns -1 if substring not found.

bool operator ==(const String& two)const- Returns boolean true, if String two matches the calling String.

string operator +(const String& two)const- Concatenates the Calling String with the parameter.

char operator [](int index)const- Returns char value at index. Exits program if an inappropriate index is given.

void operator =(const String&  two)- Changes calling String value, to argument String two value.

friend ostream& operator <<(ostream& outPut, const String& arg)- Outputs String value using standard out.

Int getCharArraySize(const char arr[])const- Returns the number of elements in a cString.

### DnaQueue class

DnaQueue()- Default constructor.

int size- number of nodes in the queue.

Node* headptr- pointer to the head of the queue.

enqueue(const String& str)- Adds a new Node with String object to the queue.

dequeuer()-Removes a node from the queue.

setSize()-Changes the size value of the queue.

setback()-Moves head of queue, to the back of the queue.

getSize()const- returns the size of the queue.

## Node class (private class inside DnaQueue)

Node()- default constructor.

Node* previous- points to previous node in queue.

Node* next- points to the next node in queue.

Gene element- A Gene object.

int permutationCount- Integer representing how many times the Gene in the node has been compared to the sequence being constructed.

## Gene class

Gene(const String & str)- Constructor for Gene class

void cleanSeq(const String& str)- Removes character that are not ‘A’, ‘C’, ‘T’, or ‘G’.

Gene operator + (const Gene& gene2)- Adds to Gene objects together, in a new Gene object.

int find(const Gene& gene2)-Returns the starting index of the first 20 matching characters.

## Main class

readfile()- Reads a file containing, a Dna sequence to be assembled by sequences stored in the file.

writeOut()- Writes the output of the program to a file.

sequence()- Constructs Dna Sequence from sequences stored in file.

## String interface

char * data- pointer to dynamic array

int _length- the number of chars in the String.

String()- default String constructor.

String(const char str[])- creates a String from a cString argument.

String(const String& toCopy)- Copy constructor.

int Length() const- Function returns the number of characters in a String.

void Clear()- Function sets String size to zero.

bool empty()const-Function returns a true if String is empty.

int find(const String & subStr, int startIndex)const- Function returns an int representing the start index of substring, in the Calling String. Returns -1 if substring not found.

bool operator ==(const String& two)const- Returns boolean true, if String two matches the calling String.

string operator +(const String& two)const- Concatenates the Calling String with the parameter.

char operator [](int index)const- Returns char value at index. Exits program if an inappropriate index is given.

void operator =(const String&  two)- Changes calling String value, to argument String two value.

friend ostream& operator <<(ostream& outPut, const String& arg)- Outputs String value using standard out.

Int getCharArraySize(const char arr[])const- Returns the number of elements in a cString.

## DnaQueue interface

DnaQueue()- Default constructor.

int size- number of nodes in the queue.

Node* headptr- pointer to the head of the queue.

enqueue(const String& str)- Adds a new Node with String object to the queue.

dequeuer()-Removes a node from the queue.

setSize()-Changes the size value of the queue.

setback()-Moves head of queue, to the back of the queue.

getSize()const- returns the size of the queue.

## Use Cases:

**Use Case name:** Run Simple Sequence on file.

**Summary:** The user Runs Simple Sequence on a file, and receives output file.

**Course of events:**

1. The user compiles the program.
2. The user then gives the argument ./a.out followed by a file name.
3. The program then reads input into currentSeq, and into a storage structure.
4. The storage structure data is compared to currentSeq.
5. Overlapping data is concatenated and stored in currentSeq.
6. Finally result of currentSeq is written to OutPut.txt.

**Alternative paths:** N/A

**Exception paths:**

1. When the user gives less than or great than two arguments, (in step 2), the program displays an error message an exits.
2. When the file to be read, (in step 3), cannot be read the program displays an error and exits.
3. When the output file cannot be written to,(in step 6), the program displays an error and exits.

**Triggers:** User wants to assemble a DNA sequence.

**Assumptions:** The Users has given a valid file to read, and assemble sequences from.

**Precondition:** String, DnaQueue, Gene, and Main classes have been compiled.

**Postcondition:** Output file containing the assembled DNA sequence.

**Test Case 4**

**Author:** Jeffrey Cocklin Date: 11-16-2016.

## Test Cases:

**<u>Test Case ID:</u> 1** **Title: Read file**

**Description:** The purpose of the test case is to insure the Program can read in input from a file.

**Tested Inputs:** ReadTest.txt

**Expected results:** Contents of the file is output to the console. In the case that file is unreadable or does not exist, an Error Message saying the file could not be read is displayed and the program exits.

**Dependences:** ReadTest.txt, Main.cpp

**Initialization:** Main.cpp is compiled, and run prior to inclusion of features 2 and 3.

**Steps:**

1. Main.cpp is compiled.
2. Two arguments are given to the commandline: ./a.out, and a filename.
3. ReadTest.txt is opened.
4. The Contents of ReadTest.txt are displayed from the console.

**Owner:** Jeffrey Cocklin

**<u>Test Case ID:</u> 2** **Title: Write file**

**Description:** The purpose of this case is to test the software's ability to write to a file.

**Tested Inputs:** ReadTest.txt

**Expected results:** The data stored in ReadTest.txt, is now stored in OutPut.txt. If either Reading or Writing functions fail, and error message will be generated and the program exits.

**Dependences:** Main.cpp, ReadTest.txt, The success of Test Case 1.

**Initialization:** Main.cpp is compiled and run after the inclusion of feature 1.

**Steps:**

1. Main.cpp is compiled
2. Two arguments are given to the commandline: ./a.out, and a filename.
3. ReadTest.txt is opened.
4. The Contents of ReadTest.txt are output to OutPut.txt.

**Owner:** Jeffrey Cocklin

**<u>Test Case ID:</u> 3** **Title: Sequence input**

**Description:** The purpose of this test case is to test the program's ability to concatenate Sequences based on overlap.

**Tested Inputs:** SeqTest1.txt, SeqTest2.txt

**Expected results:** A completed sequence output to a file OutPut.txt

**Dependences:** Success of all prior test cases. Main.cpp, String.cpp, DnaQueue.cpp, String.h, DnaQueue.h, Gene.cpp, SeqTest1.txt, SeqTest2.txt.

**Initialization:** Main.cpp, String.cpp, DnaQueue.cpp, and Gene.cpp compiled, and the program is run with an input file.

**Steps:**

1. Compile Main.cpp, String.cpp, DnaQueue.cpp, and Gene.cpp.
2. Two arguments are given to the command-line: ./a.out, and a filename.
3. The file is opened.
4. The Contents of ReadTest.txt are output to OutPut.txt.

**Owner:** Jeffrey Cocklin

## <u>Test Case ID:</u> 4 Title: Final Testing

**Description:** The purpose of this case is to ensure the program works with intended output provide by user.

**Tested Inputs:** Three input files to be given by Instructor Valafar.

**Expected results:** An output file should be generated for each of the inputs, that contains a the completely assembled target DNA sequence.

**Dependences:** Success of all prior test cases. Main.cpp, String.cpp, DnaQueue.cpp, String.h, DnaQueue.h, Gene.cpp, and the aforementioned three input files provided by instructor Valafar.

**Initialization:** Main.cpp, String.cpp, DnaQueue.cpp, and Gene.cpp compiled, and the program is run with an input file.

**Steps:**

1. Compile Main.cpp, String.cpp, DnaQueue.cpp, and Gene.cpp.
2. Two arguments are given to the command-line: ./a.out, and a filename.
3. The file is opened.
4. The Contents of the input file of sequences to be assembled are read into an instance of DnaQueue.
5. Sequence function makes comparison between two sequences looking for overlap.
6. Overlapping Strings concatenated.
7. Comparison ends after C(N,N-1) number of sequence combinations have been tried, where N equal the number of sequences to be assembled.
8. Result is written to OutPut.txt.
9. Program execution ends.

**Owner:** Jeffrey Cocklin