CMSC 202 Spring 2021

Project 3 – DNA Profiler

Assignment: Project 3 – DNA Profiler

Due Date: Thursday, April 1st @ 8:59pm on GL

Value: 80 points

1. Overview

In this project you will:

- Implement a linked-list data structure,
- Use dynamic memory allocation to create new objects,
- Practice using C++ class syntax,
- Practice object-oriented thinking.

2. Background

Forensic science is a critical element of the criminal justice system. Forensic scientists examine and analyze evidence from crime scenes and elsewhere to develop objective findings that can assist in the investigation and prosecution of perpetrators of crime or absolve an innocent person from suspicion.

Common forensic science laboratory disciplines include forensic molecular biology (DNA), forensic chemistry, trace evidence examination (hairs and fibers, paints and polymers, glass, soil, etc.), latent fingerprint examination, firearms and toolmarks examination, handwriting analysis, fire and explosives examinations, forensic toxicology, and digital evidence.

Deoxyribonucleic acid (DNA) is a molecule that carries the genetic instructions used in the growth, development, functioning and reproduction of all known living organisms. Most DNA molecules consist of two strands coiled around each other to form a double helix. The two DNA strands are termed polynucleotides since they are composed of simpler monomer units called nucleotides. The nucleotides for DNA are made up of four bases - adenine (A), guanine (G), cytosine (C), and thymine (T).

DNA sequencing is the process of determining the precise order of nucleotides within a DNA molecule. The four nucleotides (G, C, A, T) are paired. This means that if one strand of the DNA has a G, the other strand will

have a C. If one strand has an A, the other strand will have a T (and vice-versa). If you have just one of the strands, you have the leading strand. If you have both strands, each pair of nucleotides is called a base pair. Base pairs will always be (A+T, T+A, G+C, or C+G). We will be using several DNA sequences (without the pair) for this project.

This project will focus on DNA profiling. DNA profiling is the process where a specific DNA pattern, called a profile, is obtained from a person or sample of bodily tissue. We will be comparing suspect's DNA profile with evidence DNA profiles to see if there is a match.

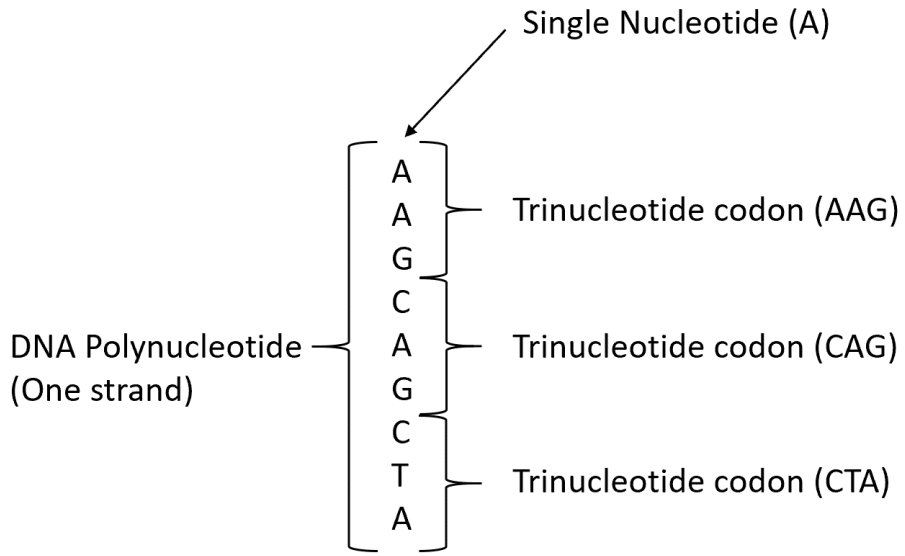

Figure 1. DNA Sequence Vocabulary

For this project, we are going to make a linked list out of our DNA sequence. There will be two or more DNA sequences for each forensic case. Each case will be comprised on types of DNA sequences: 1. Suspect's DNA sequence and 2. Evidence DNA sequence. Our primary job is to evaluate if a specific suspect sequence matches an evidence sequence.

3. Assignment Description

Your assignment is to build an application that will compare suspect and evidence DNA sequences.

1. You must use the function prototypes as outlined in the **`DNA.h and Sequencer.h`** header file. Do not edit the header files.
2. There are several input files to test including **`proj3_case1.txt`**, **`proj3_case2.txt`**, **`proj3_case3.txt`**, and **`proj3_case4.txt`**. There may be more added later to test additional edge cases. There is no defined maximum number of suspects or evidence and no defined length of the sequence itself. Do not hard code any lengths in this project. Figure 2 below shows an example input file although it could have many suspects and many evidence sequences.

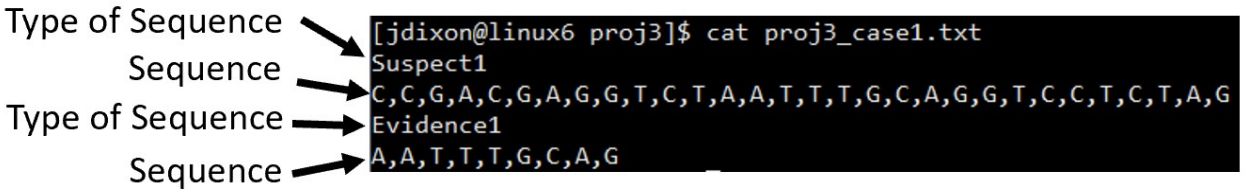

Figure 2. Input File Example

3. The **`DNA`** is the linked list in this project. The insert and getters are straightforward but the **`ReverseSequence`** and **`CompareSequence`** are challenging.
4. All user inputs will be assumed to be the correct data type. For example, if you ask the user for an integer, they will provide an integer.
5. Regardless of the sample output below, all user input must be validated. If you ask for a number between 1 and 5 with the user entering an 8, the user should be re-prompted.

4. Requirements:

Initially, you will have to use the following files **`DNA.h, Sequencer.h, makefile, proj3.cpp,`** and four input files (**`proj3_case1.txt, proj3_case2.txt, proj3_case3.txt`** and **`proj3_case4.txt`**). You can copy the files from Prof. Dixon's folder at:

`/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3`

To copy it into your project folder, just navigate to your project 3 folder in your home folder and use the command:

```
cp /afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3/* .
```

Notice the trailing period is required (it says copy it into this folder).

- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. No **`breaks`** (except in switch statements), **`continues`**, or **`exit()`**. Libraries we have learned include **`<iostream>`**, **`<fstream>`**, **`<iomanip>`**, **`<vector>`**, **`<cstdlib>`**, **`<time.h>`**, **`<cmath>`**, ~~**`<list>`**~~, and **`<string>`**. You should only use **`namespace std`**.
- You must use the function prototypes as outlined in the **`DNA.h and Sequencer.h`** header file. Do not edit the header files or **`proj1.cpp`**. Do not add functions or constants in the header files. You need to code **`DNA.cpp, and Sequencer.cpp.`**
- The **`DNA`** is the one linked list class for this project. Here are some general descriptions of each of the functions:
    - **`DNA()`** – The constructor creates a new empty linked list. **`Name`** is your choice. **`m_head`** is **`nullptr, m_tail is nullptr, and m_size = 0;`**
    - **`DNA(string name)`** – The constructor creates a new empty linked list. **`m_name`** is passed. **`m_head`** is **`nullptr, m_tail is nullptr, and m_size = 0;`**
    - **`~DNA()`** – The destructor de-allocates any dynamically allocated memory.
    - **`InsertEnd(char data)`** – Inserts a dynamically allocated node to the end of the linked list. Populated with a single nucleotide.
    - **`GetName() and GetSize()`** – Returns **`m_name`** and **`m_size`** respectively.
    - **`ReverseSequence()`** – Reverses the entire DNA sequence. Can be from **`m_suspects`** or **`m_evidence`**.
        - Hint: Do not create a new **`DNA`** (tricky to get rid of memory leaks). Look at the Linked List slides (at the end) Don't forget to populate **`m_tail`** and **`m_size`**.

- **`CompareSequence()`** – Iterates through a DNA to see if the passed DNA sequences exists in this sequence. Hint: You will almost certainly have to have a nested loop for this!

- The **`Sequencer`** class manages the application and the DNA sequences from **`Sequencer.h`**.
  - **`Sequencer ()`** – The constructor sets the file name and calls ReadFile then MainMenu.
  - **`~Sequencer()`** – The destructor deallocates each DNA sequence in both vectors.
  - **`LoadFile()`** – Opens the passenger data file passed from proj3.cpp. Input file alternates type of input (suspect or evidence) and a sequence of characters. Pay attention to the pattern – each linked list is two lines (string ->line break then string->line break). The first string is the name (Suspect1 for example) and the second string is the entire DNA sequence with each character having a comma in between them. The number of sequences is not defined. The number of nucleotides in the sequence is not defined.
  - **`MainMenu()`** – Offers the user options of displaying the sequences, reverse sequences, checking suspects, or exiting.
  - **`DisplayStrands()`** – Uses the overloaded << operator to display each strand in both vectors.
  - **`ReverseSequence()`** – Asks the user which type of sequence they would like to reverse and reverses it. Both sequences in **`m_suspect`** and **`m_evidence`** can be reversed.
  - **`CheckSuspects()`** – Compares each of the suspect's DNA sequences with each evidence DNA sequence. Indicates if they match. If there are multiple evidence sequences, it indicates if a suspect's DNA sequence includes all of the evidence DNA sequences. For our project, evidence DNA sequences are always shorter than the suspect's DNA sequence. Figure 3 shows an example of Suspect2 matching Evidence1. It is possible that an evidence DNA sequence will appear in more than one suspect DNA sequence.

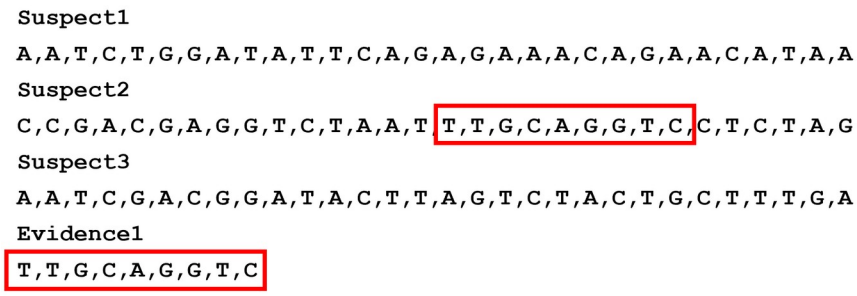

Figure 3. Showing Suspect2 matching Evidence1

5. Optional Extra Credit (up to 8pts)

Once you have completed the project using two separate vectors (**`m_suspects`** and **`m_evidence`**), you can optionally reprogram **`Sequencer.h`** and **`Sequencer.cpp`** using just one vector named **`m_dna`**. The project should have the same functionality as the base project except that there can only be one vector in **`Sequencer`** named **`m_dna`**.

As our primary goal is to help everyone get the normal project done, we are unable to offer office hours support for debugging the extra credit

To earn the extra credit for this project, you cannot change **`DNA.h`** or **`DNA.cpp`** at all. The extra credit has the same requirements as the normal project submission including no memory leaks.

To turn in this extra credit, in your project 3 submission directory, create a new folder named **`extra_credit`**. Copy all required project files to the extra credit submission directory. Extra credit cannot be turned in late.

6. Sample Input and Output

6.1. Sample Run

An additional file named **`proj3_sample.txt`** is available in

`/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3`

A normal run of the compiled code would look like this with user input highlighted in blue:

```
]$ make val1
```

```
valgrind ./proj3 proj3_case1.txt
==2155933== Memcheck, a memory error detector
==2155933== Copyright (C) 2002-2017, and GNU GPL'd, by Julian
Seward et al.
==2155933== Using Valgrind-3.16.0 and LibVEX; rerun with -h for
copyright info
==2155933== Command: ./proj3 proj3_case1.txt
==2155933==

***DNA Profiler***

Opened File
2 strands loaded.
What would you like to do?:
1. Display Strand
2. Reverse Sequence
3. Check Suspects
4. Exit
1
Suspect 1
C->C->G->A->C->G->A->G->G->T->C->T->A->A->T->T->T->G->C->A->G-
>G->T->C->C->T->C->T->A->G->END
Evidence 1
A->A->T->T->T->G->C->A->G->END
What would you like to do?:
1. Display Strand
2. Reverse Sequence
3. Check Suspects
4. Exit
3
Checking all suspects vs evidence
Suspect 1 matches Evidence 1
Suspect 1 matches ALL Evidence!
```

```
What would you like to do?:
1. Display Strand
2. Reverse Sequence
3. Check Suspects
4. Exit
2
Which type of sequence to reverse?
1. Suspect
2. Evidence
2
Done reversing Evidence 1's sequence.
What would you like to do?:
1. Display Strand
2. Reverse Sequence
3. Check Suspects
4. Exit
1
Suspect 1
C->C->G->A->C->G->A->G->G->T->C->T->A->A->T->T->T->G->C->A->G->G->T->C->C->T->C->T->A->G->END
Evidence 1
G->A->C->G->T->T->T->A->A->END
What would you like to do?:
1. Display Strand
2. Reverse Sequence
3. Check Suspects
4. Exit
3
Checking all suspects vs evidence
Suspect 1 does NOT match Evidence 1
What would you like to do?:
1. Display Strand
```

```
2. Reverse Sequence
3. Check Suspects
4. Exit
4
DNA removed from memory
Deleting Suspects
Deleting Evidence
==2155933==
==2155933== HEAP SUMMARY:
==2155933==     in use at exit: 0 bytes in 0 blocks
==2155933==   total heap usage: 49 allocs, 49 frees, 84,228
bytes allocated
==2155933==
==2155933== All heap blocks were freed -- no leaks are possible
==2155933==
==2155933== For lists of detected and suppressed errors, rerun
with: -s
==2155933== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 0 from 0)
[jdixon@linux6 proj3]$
```

Here is a longer example run where there are six suspect DNA sequences and two evidence DNA sequences. It shows what happens when

```
make val4
valgrind ./proj3 proj3_case4.txt
==2156330== Memcheck, a memory error detector
==2156330== Copyright (C) 2002-2017, and GNU GPL'd, by Julian
Seward et al.
==2156330== Using Valgrind-3.16.0 and LibVEX; rerun with -h for
copyright info
==2156330== Command: ./proj3 proj3_case4.txt
==2156330==

***DNA Profiler***
```

```
Opened File
8 strands loaded.
What would you like to do?:
1. Display Strand
2. Reverse Sequence
3. Check Suspects
4. Exit
1
Suspect 1
G->T->A->A->T->T->A->A->T->G->T->T->T->T->A->G->T->A->A->T->A-
>C->G->T->A->T->T->A->G->A->C->A->C->G->C->T->G->G->G->C->G->A-
>C->C->G->G->A->G->A->C->T->T->G->C->C->T->T->C->C->C->A->C->T-
>C->A->C->C->G->T->A->T->A->T->T->G->T->C->T->C->A->C->T->T->C-
>C->C->T->C->G->G->C->A->G->T->T->G->G->A->A->T->G->T->G->C->A-
>T->C->G->C->G->T->C->C->G->C->A->G->G->T->T->T->A->T->C->G->T-
>A->C->T->C->T->C->A->T->A->T->G->C->A->C->A->T->C->A->A->A->G-
>A->G->A->T->T->T->G->T->G->A->C->A->A->T->C->G->T->C->T->T->A-
>A->A->T->C->T->T->T->C->T->G->T->C->A->T->C->T->A->T->T->C->T-
>C->T->C->A->T->A->T->G->T->G->T->G->A->T->T->C->A->A->C->C->C-
>A->T->G->G->G->T->A->A->A->G->C->T->A->T->T->C->C->T->A->G->T-
>A->G->G->C->C->G->G->G->C->T->A->C->C->C->T->G->T->T->C->A->A-
>C->G->C->A->G->A->C->C->G->G->A->A->C->C->A->A->G->A->C->T->C-
>T->T->C->T->T->C->C->T->T->G->G->A->A->T->C->C->T->A->T->C->A-
>G->C->A->A->A->A->T->A->A->END
Suspect 2
A->C->C->A->A->A->T->G->C->G->T->G->T->T->A->G->G->T->A->G->C-
>G->G->T->A->T->C->A->T->A->C->A->C->T->C->T->T->C->T->A->C->T-
>A->T->C->G->G->T->T->G->T->G->C->A->A->G->T->G->G->G->T->C->T-
>G->T->G->G->C->C->T->G->G->A->T->A->G->C->A->G->A->C->G->G->T-
>A->G->T->G->C->T->A->C->T->C->C->G->T->G->G->G->T->A->G->T->G-
>T->C->T->T->C->A->A->C->C->G->G->T->G->C->C->A->T->G->C->G->G-
>C->A->T->T->G->T->A->A->T->C->A->A->A->T->T->C->C->C->C->A->T-
>C->C->G->A->A->T->A->A->A->C->T->A->G->T->A->T->G->T->T->T->G-
>G->G->T->G->C->A->A->C->T->T->G->C->G->T->A->G->T->A->C->T->A-
>C->T->T->A->G->C->C->A->C->A->C->T->C->G->G->A->G->A->T->A->T-
>G->A->G->G->G->T->G->A->G->C->G->T->A->A->A->T->C->A->G->A->T-
>C->G->T->C->C->A->A->A->G->G->A->A->A->G->G->G->G->A->C->A->G-
>C->A->A->A->T->T->G->A->T->G->C->G->A->G->C->G->T->G->T->G->C-
>G->G->G->T->T->A->C->C->A->C->C->C->G->G->G->G->A->T->A->G->G-
```

```
>A->T->A->T->T->A->T->A->G->END

Suspect 3

T->T->A->G->A->G->A->G->T->C->T->A->G->C->T->A->G->A->T->A->T-
>C->C->T->C->T->T->T->C->A->G->A->T->G->A->T->A->C->T->G->G->A-
>T->T->C->G->C->C->T->T->T->A->T->T->C->C->C->C->A->C->C->A->T-
>T->C->T->G->C->T->A->C->A->A->C->G->T->C->A->G->T->T->G->T->T-
>A->C->G->G->G->C->T->T->A->C->A->A->A->C->C->A->A->G->A->T->C-
>A->A->G->C->A->A->G->T->A->T->T->A->T->G->T->G->T->A->T->T->C-
>G->G->G->G->G->C->C->T->T->G->A->C->C->G->G->G->G->C->A->C->A-
>T->T->C->C->T->C->G->C->C->A->A->C->T->T->G->A->T->T->C->G->C-
>G->A->T->G->T->A->T->A->T->T->T->G->G->T->T->A->T->T->G->T->C-
>C->A->G->G->T->C->G->C->T->G->A->G->A->A->G->G->A->C->T->G->G-
>T->G->C->T->C->G->C->G->G->C->T->C->C->C->A->T->C->A->A->A->G-
>G->T->C->T->G->T->A->T->C->A->T->A->G->A->C->A->G->A->T->T->T-
>G->G->A->C->C->T->C->T->C->G->G->T->T->C->C->A->A->G->C->A->G-
>T->A->T->C->T->G->T->T->G->G->T->C->G->T->A->A->C->T->C->A->A-
>A->G->T->T->G->G->T->G->A->END

Suspect 4

C->T->A->T->G->T->A->A->A->C->G->A->G->T->C->G->T->C->A->G->T-
>C->G->T->A->T->C->C->A->A->T->C->C->C->C->T->C->G->A->G->C->T-
>C->A->C->G->A->G->G->T->G->C->G->A->G->A->G->A->T->T->T->C->A-
>G->T->G->C->A->G->A->G->C->T->G->C->A->A->T->T->G->C->A->A->T-
>C->A->T->A->G->T->T->T->G->T->A->C->T->C->A->A->C->C->C->A->G-
>A->A->G->T->C->T->T->T->C->C->C->T->G->A->G->T->T->T->C->C->G-
>C->C->T->A->A->C->A->C->A->A->G->G->C->G->C->C->T->G->G->C->A-
>T->T->T->C->C->C->A->A->A->C->G->C->G->T->G->A->T->A->C->T->C-
>G->T->C->G->G->A->C->G->C->C->G->A->C->T->C->C->C->A->C->T->A-
>A->A->G->T->C->A->G->T->A->T->T->A->T->A->T->G->C->C->T->G->G-
>G->T->T->C->A->A->G->A->T->G->G->A->A->G->A->A->G->A->C->T->A-
>A->T->C->G->G->C->C->T->C->T->C->G->A->T->G->C->G->G->T->G->C-
>A->A->T->A->G->C->G->A->A->C->A->A->T->G->G->C->A->A->C->T->A-
>T->G->C->T->C->G->T->G->T->C->C->C->A->A->G->C->C->G->A->A->G-
>C->A->T->A->C->C->T->A->A->END

Suspect 5

T->C->A->G->A->G->T->T->C->A->G->A->T->C->C->T->T->G->C->G->T-
>T->T->T->T->T->G->C->T->T->A->C->C->A->C->C->G->A->C->C->A->T-
>C->G->T->A->C->C->C->G->A->T->T->T->G->T->T->A->T->A->T->G->T-
>T->G->G->A->G->C->A->A->A->A->C->A->C->T->T->A->A->G->G->C->C-
>G->G->T->C->T->T->A->T->T->A->G->A->A->C->G->A->C->A->C->G->C-
>G->G->G->T->G->T->A->C->A->C->A->A->C->A->G->G->T->G->C->A->C-
>G->C->C->G->C->C->T->A->C->C->T->T->A->T->C->A->C->A->A->C->G-
>T->T->C->C->T->T->T->C->A->G->G->T->A->G->C->T->T->C->G->G->C-
>G->C->T->C->T->C->A->C->T->C->T->G->C->T->A->T->T->A->T->G->T-
```

```
>C->C->C->T->C->T->T->C->C->G->A->C->A->C->G->T->T->G->G->A->A-
>A->T->T->C->C->G->A->G->T->G->C->A->G->T->T->A->A->A->G->A->G-
>C->A->A->C->G->T->T->T->A->A->C->T->A->T->G->G->C->C->G->T->T-
>G->A->C->G->G->A->T->A->T->G->T->C->A->T->T->C->C->G->C->C->T-
>G->T->G->C->A->C->A->A->G->G->C->C->C->A->G->T->T->A->A->T->G-
>T->C->A->C->G->C->T->A->G->END

Suspect 6

T->G->T->A->T->T->G->A->T->C->T->A->T->A->C->A->G->C->G->C->G-
>C->G->A->A->C->A->C->G->C->G->T->G->A->A->G->C->T->C->C->A->T-
>A->G->C->G->A->A->C->A->A->T->G->G->C->A->A->A->C->T->T->C->A-
>G->C->A->C->T->C->G->G->A->C->G->T->A->C->C->A->T->G->T->G->T-
>A->G->G->G->C->C->A->A->G->C->T->A->A->G->G->T->T->A->T->A->T-
>C->T->T->A->G->G->T->G->C->C->G->T->A->T->A->A->G->G->C->G->A-
>G->C->C->G->T->T->T->G->G->C->C->A->A->A->C->C->T->T->T->G->G-
>G->G->C->T->G->T->A->A->A->A->G->T->A->T->C->A->C->A->C->G->T-
>G->G->T->T->C->T->C->T->T->G->G->C->A->C->T->C->G->A->A->G->T-
>A->C->G->G->C->A->C->C->G->C->T->A->C->A->G->G->A->A->A->A->C-
>C->A->T->A->A->A->T->C->T->C->G->G->G->C->G->C->T->A->T->C->A-
>G->C->C->C->G->A->C->C->G->T->G->G->C->T->G->A->C->A->T->T->G-
>T->C->C->T->C->A->G->T->G->A->A->T->C->T->A->A->T->A->G->G->T-
>T->G->C->A->A->G->A->A->T->G->T->C->G->T->C->A->G->T->C->G->T-
>A->T->C->C->A->A->T->G->A->END

Evidence 1

A->A->T->A->G->C->G->A->A->C->A->A->T->G->G->C->A->A->END

Evidence 2

G->T->C->G->T->C->A->G->T->C->G->T->A->T->C->C->A->A->END

What would you like to do?:

1. Display Strand

2. Reverse Sequence

3. Check Suspects

4. Exit

3

Checking all suspects vs evidence

Suspect 1 does NOT match Evidence 1

Suspect 1 does NOT match Evidence 2

Suspect 2 does NOT match Evidence 1

Suspect 2 does NOT match Evidence 2

Suspect 3 does NOT match Evidence 1
```

```
Suspect 3 does NOT match Evidence 2
Suspect 4 matches Evidence 1
Suspect 4 matches Evidence 2
Suspect 4 matches ALL Evidence!
Suspect 5 does NOT match Evidence 1
Suspect 5 does NOT match Evidence 2
Suspect 6 does NOT match Evidence 1
Suspect 6 matches Evidence 2
What would you like to do?:
1. Display Strand
2. Reverse Sequence
3. Check Suspects
4. Exit
4
DNA removed from memory
Deleting Suspects
Deleting Evidence
==2156330==
==2156330== HEAP SUMMARY:
==2156330==     in use at exit: 0 bytes in 0 blocks
==2156330==   total heap usage: 1,874 allocs, 1,874 frees,
114,278 bytes allocated
==2156330==
==2156330== All heap blocks were freed -- no leaks are possible
==2156330==
==2156330== For lists of detected and suppressed errors, rerun
with: -s
==2156330== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 0 from 0)
```

7. Compiling and Running

Because we are using a significant amount of dynamic memory for this project, you are required to manage any memory leaks that might be created.

For a linked list, this is most commonly related to the dynamically allocated nodes. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

One way to test to make sure that you have successfully removed any of the memory leaks is to use the **`valgrind`** command.

Since this project makes extensive use of dynamic memory, it is important that you test your program for memory leaks using **`valgrind`**:

```
valgrind ./proj3 proj3_case1.txt
```

Note: If you accidently use **`valgrind make run`**, you may end up with some memory that is still reachable. Do not test this – test using the command above where you include the input file. The **`makefile`** should include **`make val`** (which is ok).

If you have no memory leaks, you should see output like the following:

```
==5606==
==5606== HEAP SUMMARY:
==5606==     in use at exit: 0 bytes in 0 blocks
==5606==   total heap usage: 87 allocs, 87 frees, 10,684 bytes
allocated
==5606==
==5606== All heap blocks were freed -- no leaks are possible
==5606==
==5606== For counts of detected and suppressed errors, rerun
with: -v
==5606== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6
from 6)
```

The important part is “in use at exit: 0 bytes 0 blocks,” which tells me all the dynamic memory was deleted before the program exited. If you see anything other than "0 bytes 0 blocks" there is probably an error in one of your destructors. We will evaluate this as part of the grading for this project.

Additional information on **`valgrind`** can be found here:
http://valgrind.org/docs/manual/quick-start.html

Once you have compiled using your **`makefile`**, enter the command **`./proj3`** to run your program. You can use **`make val`** to test each of the

input files using **`valgrind`** (do NOT use **`valgrind make run`**!). They have differing sizes. If your executable is not **`proj3`**, you will lose points. It should look like the sample output provided above.

8. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

For this project, you should submit these files to the **`proj3`** subdirectory:

`proj3.cpp` — should be unchanged.

`DNA.h` — should be unchanged.

`DNA.cpp` – should include your implementations of the class functions.

`Sequencer.h` — should be unchanged.

`Sequencer.cpp` – should include your implementations of the class functions.

As you should have already set up your symbolic link for this class, you can just copy your files listed above to the submission folder. You should turn in all files for this project.

a. cd to your project 3 folder. An example might be cd
```
~/202/projects/proj3
```

b.
```
cp proj3.cpp DNA.h DNA.cpp Sequencer.cpp Sequencer.h
~/cs202proj/proj3
```

You can check to make sure that your files were successfully copied over to the submission directory by entering the command

```
ls ~/cs202proj/proj3
```

You can check that your program compiles and runs in the **`proj3`** directory, but please clean up any **`.o`** and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here. Uploading or generation of any **`.gch or vgcore*`** files in your submit directory will result in a severe penalty.

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer

copy the files into the proj3 folder, it is because the due date has passed. You should be able to see your proj3 files, but you can no longer edit or copy the files in to your proj3 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to **`~/cs202proj/proj3-late1`**
- If it is 24-48 hours late, copy your files to **`~/cs202proj/proj3-late2`**
- If it is after 48 hours late, it is too late to be submitted.