CMSC 202 Spring 2023

Project 3 – Transcription and Translation

Assignment: Project 3 – Transcription and Translation

Due Date: Tuesday, April 4th @ 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

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).

Transcription and translation are processes a cell uses to make all of the proteins the body needs to function from information stored in the sequence of bases in DNA. Transcription is the first step in gene expression, in which information from a gene is used to construct a functional product such as a protein. The goal of transcription is to make an mRNA copy of a gene's DNA sequence. For a protein-coding gene, the mRNA copy, or transcript, carries the information needed to build a polypeptide (protein or protein subunit).

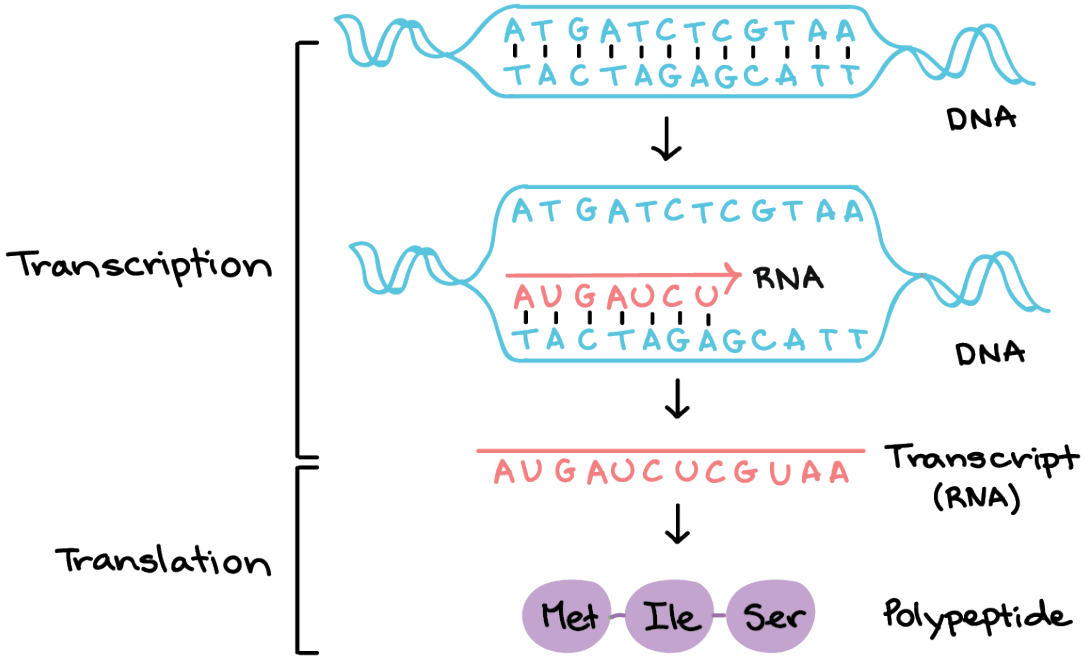

Figure 1. Transcription and Translation

During transcription, a piece of DNA that codes for a specific gene is copied into messenger RNA (mRNA) in the nucleus of the cell. The mRNA then carries the genetic information from the DNA to the cytoplasm, where translation occurs. When DNA is transcribed into mRNA, T (which is changed to U in mRNA) complements A, A complements U, C complements G, and G complements C.

During translation, proteins are made using the information stored in the mRNA sequence. The mRNA attaches to a structure called a ribosome that can read the genetic information. As the mRNA passes through the ribosome, another type of RNA called transfer RNA (tRNA) carries a protein building block called an amino acid to the ribosome. The tRNA carrying the amino acid binds to a matching sequence in the mRNA. As each tRNA binds to the mRNA strand, the amino acid it carried joins with the other amino acids to form a chain of amino acids. Once all the amino acids coded for in the piece of mRNA have been linked, the completed protein is released from the ribosome.

Here is an example for how transcription and translation works: https://learn.genetics.utah.edu/content/basics/transcribe/

For this project, we will be loading in a single strand of DNA (picturing that the DNA has already been spliced) at a time. The DNA will then be transcribed

into mRNA. Finally, the mRNA will be converted into polypeptides using translation. Every three nucleotides, called a codon, will translate into one amino acid as shown in Figure 2 below.

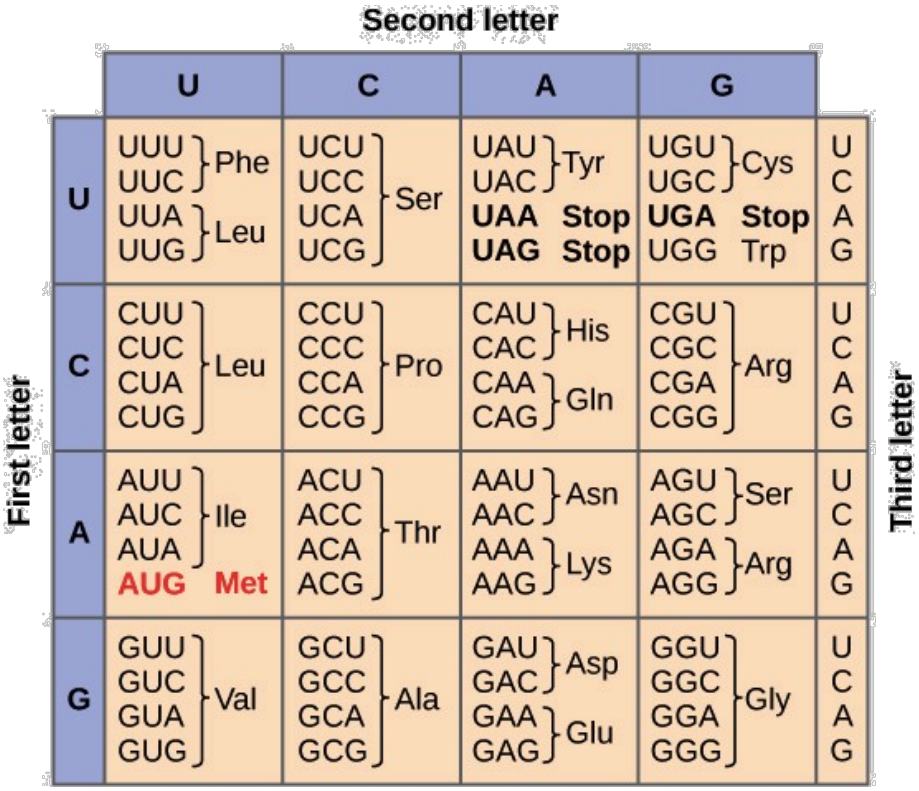

Figure 2. Mapping RNA to Amino Acids

3. Assignment Description

Your assignment is to build an application that model transcription and translation.

1. You must use the function prototypes as outlined in the **`Strand.h and Sequencer.h`** header file. Do not edit the header files.
2. There are several input files to test including **`proj3_data1.txt`**, **`proj3_data2.txt`**, **`proj3_data3.txt`**, and **`proj3_data4.txt`.** There is no defined maximum number of nucleotides in a file (although they will always be a multiple of 3) and no defined length of the strand itself. Do not hard code any

lengths in this project. Figure 3 below shows an example input file with three defined strands.

```
Glow Worm,T,A,C,C,T,A,C,T,T,A,C,G,A,A,C,A,C,T
Eastern Diamondback Rattlesnake,T,A,C,G,C,C,T,A,G,G,C,G,G,T,A,A,T,C
Domestic Cat,T,A,C,G,G,G,T,A,G,C,T,A,G,C,C,A,T,T
```

Figure 3. Input File Example

3. The **DNA** and the **mRNA** both use a linked list in this project. The insert and getters are straightforward, but the **ReverseSequence** function is 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 **Strand.h, Sequencer.h, makefile, proj3.cpp,** and four input files (**proj3_data1.txt, proj3_data2.txt, proj3_data3.txt** and **proj3_data4.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 **Strand.h and Sequencer.h** header file. Do not edit the header files or **proj3.cpp**.

Do not add functions or constants in the header files. You need to code **`Strand.cpp, and Sequencer.cpp.`**

- The **`Strand`** is the one linked list class for this project. Here are some general descriptions of each of the functions:
    - **`Strand()`** – The constructor creates a new empty linked list. **`Name`** is your choice. **`m_head`** is **`nullptr, m_tail is nullptr, and m_size = 0;`**
    - **`Strand(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;`**
    - **`~Strand()`** – 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 strand.

      Can reverse a strand from **`m_DNA`** or **`m_mRNA`**.
        - Hint: Do not create a new **`Strand`** (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`**.
- The **`Sequencer`** class manages the application and the DNA and mRNA strands from **`Sequencer.h`**.
    - **`Sequencer ()`** – The constructor sets the file name that is passed to it
    - **`~Sequencer()`** – The destructor deallocates each DNA sequence in both vectors.
    - **`LoadFile()`** – Opens **`m_fileName`** from the Sequencer constructor (passed in from **`proj3.cpp`**). Each line will include the name of the source of the DNA and a sequence of characters. The first string is the name (for example John Doe) 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.

- **`StartSequencing()`** – Calls ReadFile and continually calls MainMenu until the user quits the application.
- **`MainMenu()`** – Offers the user options of displaying the strands, reverse sequences, transcribing strands, translating a strand, or exiting. Returns 5 if the user quits, else 0.
- **`DisplayStrands()`** – Uses the overloaded << operator to display each strand in both the **`m_DNA`** and the **`m_mRNA`** vectors. Includes the number of the type of strand (for example, DNA 1) and the name of the strand (for example, John Doe).
- **`ChooseDNA()`** – Allows the user to choose a specific DNA strand to reverse.
- **`ChooseMRNA()`** – Allows the user to choose a specific mRNA strand to reverse or translate
- **`ReverseSequence()`** – Asks the user which type of strand they would like to reverse and reverses it. Both strands in **`m_DNA`** and **`m_mRNA`** can be reversed. Calls either **`ChooseDNA`** or **`ChooseMRNA`** and then reverses the strand.
- **`Transcribe()`** – Iterates through each strand in DNA and creates a new mRNA strand using complementary nucleotides. To transcript DNA into mRNA, Uracil (U) replaces Thymine (T). So, G ↔ C, A → U, and T → A. When the entire strand has been transcribed, it puts the new, populated strand into **`m_mRNA`** vector. This can be done for any number of strands at once.
- **`Translate()`** – Asks the user which specific strand of mRNA to use (Calls **`ChooseMRNA`**). Then iterates through that strand and for every three nodes, calls Convert. Displays each of the amino acids from that strand. Does NOT store this data anywhere in the class.
- **`Convert()`** – This provided function, expects a codon (three letters from mRNA) and returns the corresponding amino acid. Figure 2 shows all of the conversions available from mRNA to amino acids.

5. Sample Input and Output

5.1. 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:

```
[jdixon@linux4 proj3]$ make run1
./proj3 proj3_data1.txt

***Transcription and Translation***

Opened File
1 strands loaded.
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
1
DNA 1
***John Doe***
T->A->C->G->C->T->A->T->T->END
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
3
```

```
1 strands of DNA successfully transcribed into new mRNA strands
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
1
DNA 1
***John Doe***
T->A->C->G->C->T->A->T->T->END
mRNA 1
***John Doe***
A->U->G->C->G->A->U->A->A->END
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
4
***John Doe***
AUG->Methionine (START)
CGA->Arginine
UAA->Stop
Done translating mRNA 1's strand.
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
```

```
5
Exiting program
Deleting DNA Strands
Deleting mRNA Strands
```

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

```
[jdixon@linux4 proj3]$ make val3
valgrind ./proj3 proj3_data3.txt
==2427965== Memcheck, a memory error detector
==2427965== Copyright (C) 2002-2022, and GNU GPL'd, by Julian
Seward et al.
==2427965== Using Valgrind-3.19.0 and LibVEX; rerun with -h for
copyright info
==2427965== Command: ./proj3 proj3_data3.txt
==2427965==

***Transcription and Translation***

Opened File
3 strands loaded.
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
1
DNA 1
***Glow Worm***
T->A->C->C->T->A->C->T->T->A->C->G->A->A->C->A->C->T->END
DNA 2
***Eastern Diamondback Rattlesnake***
```

```
T->A->C->G->C->C->T->A->G->G->C->G->G->T->A->A->T->C->END
DNA 3
***Domestic Cat***
T->A->C->G->G->G->T->A->G->C->T->A->G->C->C->A->T->T->END
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
2
Which type of strand to reverse?
1. DNA
2. mRNA
1
Which strand to work with?
Choose 1 - 3
3
Done reversing DNA 3's strand.
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
2
Which type of strand to reverse?
1. DNA
2. mRNA
2
No mRNA to reverse; transcribe first
What would you like to do?:
```

```
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
2
Which type of strand to reverse?
1. DNA
2. mRNA
1
Which strand to work with?
Choose 1 - 3
3
Done reversing DNA 3's strand.
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
4
No mRNA to translate; transcribe first
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
3
3 strands of DNA successfully transcribed into new mRNA strands
What would you like to do?:
1. Display Strands
```

```
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
1
DNA 1
***Glow Worm***
T->A->C->C->T->A->C->T->T->A->C->G->A->A->C->A->C->T->END
DNA 2
***Eastern Diamondback Rattlesnake***
T->A->C->G->C->C->T->A->G->G->C->G->G->T->A->A->T->C->END
DNA 3
***Domestic Cat***
T->A->C->G->G->G->T->A->G->C->T->A->G->C->C->A->T->T->END
mRNA 1
***Glow Worm***
A->U->G->G->A->U->G->A->A->U->G->C->U->U->G->U->G->A->END
mRNA 2
***Eastern Diamondback Rattlesnake***
A->U->G->C->G->G->A->U->C->C->G->C->C->A->U->U->A->G->END
mRNA 3
***Domestic Cat***
A->U->G->C->C->C->A->U->C->G->A->U->C->G->G->U->A->A->END
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
4
Which strand to work with?
Choose 1 - 3
```

```
3
***Domestic Cat***
AUG->Methionine (START)
CCC->Proline
AUC->Isoleucine
GAU->Aspartic acid
CGG->Arginine
UAA->Stop
Done translating mRNA 3's strand.
What would you like to do?:
1. Display Strands
2. Reverse Strand
3. Transcribe DNA to mRNA
4. Translate mRNA to Amino Acids
5. Exit
5
Exiting program
Deleting DNA Strands
Deleting mRNA Strands
==2427965==
==2427965== HEAP SUMMARY:
==2427965==     in use at exit: 0 bytes in 0 blocks
==2427965==   total heap usage: 135 allocs, 135 frees, 85,903 bytes allocated
==2427965==
==2427965== All heap blocks were freed -- no leaks are possible
==2427965==
==2427965== For lists of detected and suppressed errors, rerun with: -s
==2427965== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

6. 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, memory leaks are most commonly caused by the

dynamically allocated nodes. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

You will need to write your own makefile for this project. Don't forget to write some additional macros such as "make run1" or "make val1" and maybe some macros to help you submit. Use project 2's makefile as a template.

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_data1.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.

7. 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.

`Strand.h` — should be unchanged.

`Strand.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 Strand.h Strand.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.