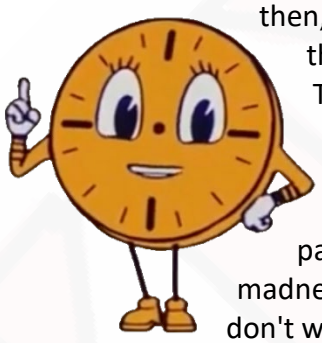
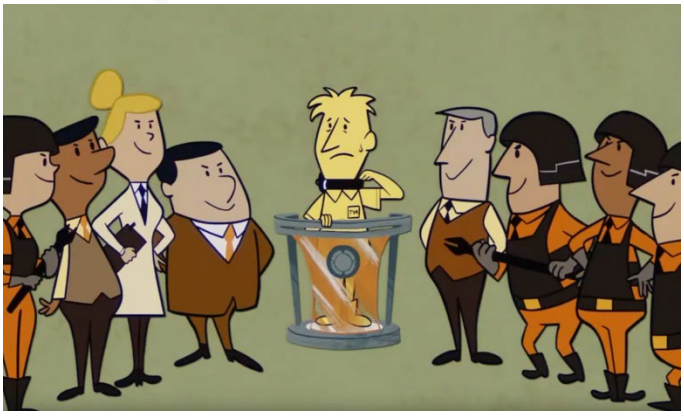
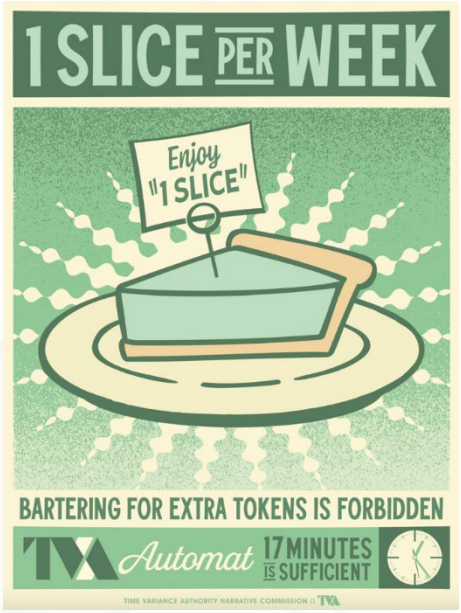
Assignment 8 Overview: The Time Variance Authority

Hey there! You're probably saying, "This is a mistake. I shouldn't even be here." Welcome to the Time Variance Authority. I'm Miss Minutes, and it's my job to catch you up before you stand trial for your crimes. So, let's not waste another minute. Settle in, sharpen your pencils, and check this out. Long ago, there was a vast Multiversal war. Countless unique timelines battled each other for supremacy, nearly resulting in the total destruction of, well, everything. But then, the all-knowing Timekeepers emerged, bringing peace by re-organizing the Multiverse into a single timeline, the Sacred Timeline. Now, the Timekeepers protect and preserve the proper flow of time for everyone and everything. But sometimes, people like you veer off the path the Timekeepers created. We call those variants. Maybe you started an uprising, or were just late for class; whatever it was, stepping off your path created a Nexus event, which, left unchecked, could branch off into madness, leading to another Multiversal war! But don't worry, to make sure that doesn't happen, the Timekeepers created the TVA and all its incredible workers! The TVA has stepped in to fix your mistake, and set time back on its predetermined path. Now that your actions have left you without a place on the timeline, you must stand trial for your offenses. So sit tight and we'll get you in front of a judge in no time! Just make sure you have your ticket, and you'll be seen by the next available attendant. TVA - For all time, always!





Hi Variant! I am Mobius, and I have convinced the Judge to spare you from immediate pruning so you can help us out. I know you have strong programming skills and I think you are the right person for the job! A variant has been taking out our minutemen and the remaining ones are having a hard time pruning all the branches caused by nexus events. So, I want you to write a program that can automatically prune the branches as they appear. Plus, with all the time saved I can go eat some more key lime pie!



In this assignment you will create a singly linked list simulating the sacred timeline. After that you will then create nexus events with a function. You will then create a function to find and prune these nexus events you created restoring the sacred timeline for all time. Always. In addition to this you will create 2 print functions to print the status of the sacred timeline at different stages over the above events both as a text-based representation and as an ascii-visual representation of the timeline and the branches. Oh and, of course, you will do this all while keeping the TVA memory leak free. We don't want to overload the temporal loop after all with any leaks! **-OB**

```
___________   ____
| /|     \   \ /   / \
|/ |     |\   Y \/    \
   |     | \    /\     \
   |____|   \_/_/\____\
 FOR ALL TIME.  ALWAYS.

==9014==
==9014== HEAP SUMMARY:
==9014==     in use at exit: 0 bytes in 0 blocks
==9014==   total heap usage: 92 allocs, 92 frees, 76,564 bytes allocated
==9014==
==9014== All heap blocks were freed -- no leaks are possible
==9014==
==9014== For counts of detected and suppressed errors, rerun with: -v
==9014== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Overview of files and deliverables:

The assignment consists of one file. main.cpp; to compile you simply do g++ main.cpp. You are given a skeleton main.cpp that you will need to complete to run your program. You are only to code in the sections that are mentioned in the documentation of the skeleton file. Do not modify any existing code including the main function.

Overview of main.cpp:

This file should contain 2 classes called **node** and **Timeline** along with the given **main function**. The node class should contain an integer `year`, an integer `branches`, and a node pointer `next`. It should contain two constructors. A default constructor and a constructor with an integer parameter to initialize the year class member variable. In the case of the default constructor year should initialize to 0. In both constructors, branches should initialize to 0 and next should be initialized to `nullptr`. Everything (variable and constructors) is set to public.

In the Timeline class you have two private variables: integer `size` and node pointer `head`. **No other variables are allowed in either class.** All functions in the Timeline class are set to public. The functions in Timeline class should be as follows:

Constructor

This function should have int start and int size as its parameters using as default parameters both being set to 0 if no arguments ae passed in. The start parameter represents the starting year of the sacred timeline, for example, 1970. The size represents how many years are to be in the sacred timeline. So, if this value is 50 that means the timeline will contain 1970, 1971 … up to 2019. The function should create the linked list by creating a year per value up to the size given in. Make sure you set the head pointer of the class to point to the first node and that the last node keeps a `nullptr` in the next variable to indicate it is the end of the timeline.

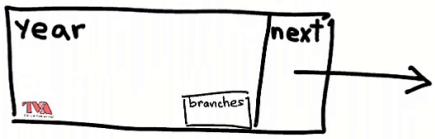
createNexusEvent

This function will create the nexus event. It takes in an integer parameter called amount that will default to 1 if no parameter is passed in. The value represents how many nexus events to create in the sacred timeline. The function is an integer value returning function that returns the number of nexus events created. If the linked list is empty when it is called the function returns a 0 and does nothing. Otherwise, it will randomly find a location in the timeline to make a branch of a random size. Please take careful note that in the skeleton code I give you the first few lines of this function. This is because I want to make sure that your code's randomization matches the one used in the grading software. There are two calls to `rand()` in the entire codebase and both are found in this function. The first one is to select a year per nexus event and the second one is to decide how big the branch is. To decide the size of the branch I take the starting year and add the size of the timeline minus the year that was randomly picked for

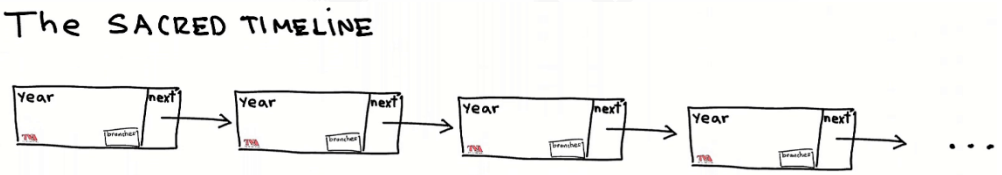
the nexus event. This ensures that the branch is not bigger than the timeline. However, after further testing I didn't like how big the branches were, as it did not seem canon, so I divided this number by 4 which ensures the branches don't look too big and overlap. Again, this code is given to you so don't mess with it but do understand it. The rest of the function is up to you to use these two values to branch the timeline and don't forget to delete the year after the nexus event since you replace it with a non-pointer type in `next[0]`.

Did the last sentence confuse you? That's alright because this is the most important part of the assignment so let me hand it over to, HE WHO REMAINS so he can explain how we are able to convert a singly linked list to essentially point to two or more nodes (the original timeline plus the branches):

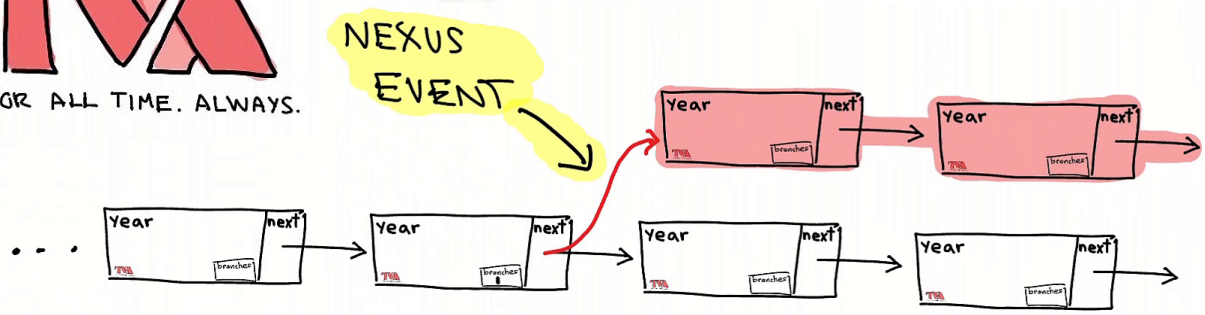
Hi! I am He Who Remains! I created the TVA and, in a way, led you to where you are now, here, in front of me! Now I want to take a moment to emphasize what Miss Minutes explained about the sacred timeline by going over some technical details on how it all works. As you learned already, each year in the timeline is represented by a Node. Like this!



These nodes together make the sacred timeline.



It's important to note that each next pointer is of node type and can only point to one other node. So, Year 1893 can only point to YEAR 1894. After all, it is a singly linked list. Also, by default all nodes have a 0 in number of branches since they do not have any branches, and shouldn't, since it is your job to prune them after all. Now when a Nexus event happens the timeline looks like this:



But you may be asking yourself, "how in the world is a single pointer, `next`, pointing to both the sacred timeline, and the new branch!?"

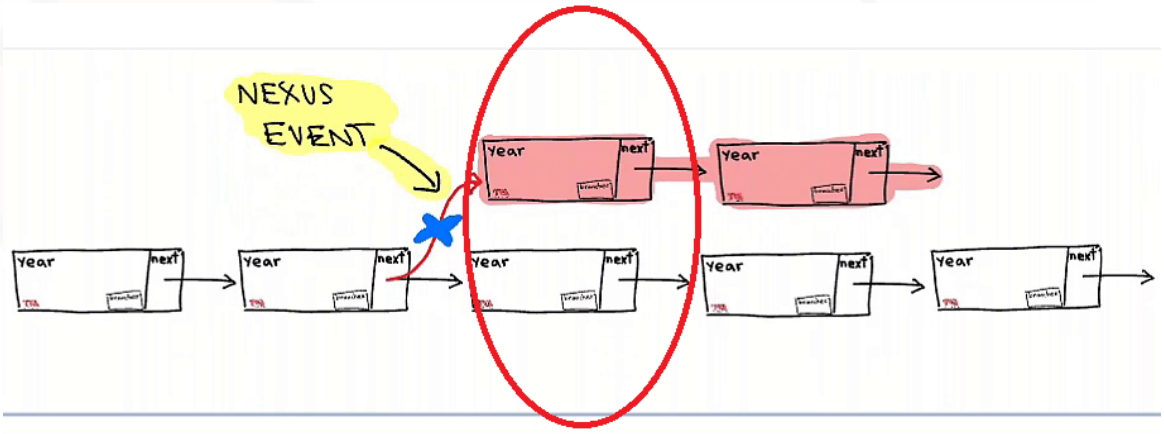
Well, a Variant of myself lived on Earth in the 31st Century. He was a scientist, and he discovered that you could take a pointer of a datatype and do two things to it. You can either make it point to an object dynamic, or not, such as:

```
int a;                  //random int a
int *ptr;               //random pointer of int type
ptr = new int;          //pointer points to new int
 or
int *ptr = &a;          //pointer points to a (alias)
```

or you can also use it to point to a dynamically allocated 1-Dimensional array such as:

```
int *ptr;               //random pointer of int type
ptr = new int[5];       //pointer points to new int array
```

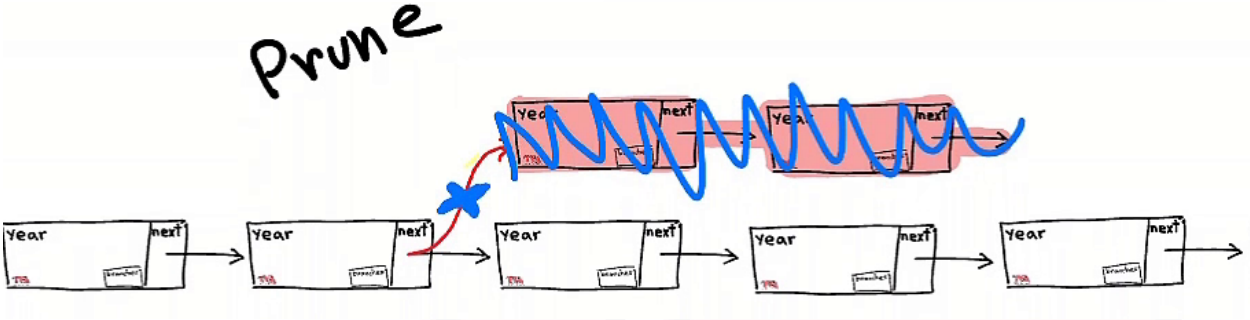
In a linked list we typically use the first of the two usages by making it point to the next node. But what if instead of making it point to the next node, we dynamically allocate a 1-Dimensional array of node type of size 2. In the first entry we copy over the next node of the next year, and in the second entry we put the branching timeline which happens to be a node of the current year. Bingo! That is how we achieve our goal! The only catch is we must copy the contents of the next dynamic node and then remove it to prevent memory leaks, similarly the first node of the branch is in our array, so they are a little different than the free nodes that follow after them, and the code looks different, but outside of that the rest of the nodes are single dynamic nodes like before!



It's the two nodes in the red circle that are the `node[2]` array. You will use the member dot operator instead of the member arrow operator to access elements in them. This is normal. Because C++ has no way of knowing what the pointer points to, you need to use the fact that the branch integer is increased to 1 when you create these branches, so you know to read it as an array. Any other node outside of that circle is done like usual with `new node(year)`.

Next finding these nexus events is as easy as finding where the integer for branches is not 0. Then you know how to interpret this as an array and reverse the process to prune them. To

reverse the process, you need to create a new node for the following year from the branch and copy over the contents of index [0] from the array. After that de-allocate all branched nodes and then de-allocate the array of size 2 and finally relink the next pointer to that node you made which because you copied the contents should already point to the next year in the sacred timeline.



That's all there is to it. It's a little tricky and I recommend you test it out with a normal linked list before you go and do your assignment but doing this little hack allows you to have branched timelines! The whole function should be about 30-35 lines of code.

Print

This function prints out the sacred timeline and any branches that may exist. See sample output for how it should be formatted. It should be about 20 lines of code.

vprint

This function is a variant of the prints out the sacred timeline and any branches that may exist. The difference is this does more of a visual representation by printing - ~ \ / symbols to sort of draw what the timeline looks like to make it easier to visualize than the raw text of the other print function. This will probably be your longest function at about 70-80 lines but that is mostly because part of the code is duplicated for drawing above the timeline or beneath. The documentation on the skeleton code provides more detail but basically you want to draw each branch on opposite sides starting with above. Don't worry if two branches are on top of each other and you can only see one of them. The print function is the one that gives us accurate representation. This is just a visual aid but does match the output for grading purposes.

Prune

This function prunes the first branch it sees from left (past) to right (future) and returns the location it pruned for logging purposes. As explained by He Who Remains, it is mostly the reverse of creating the nexus event, so the TVA recommends you refer to that section for further details. It should be about 35 lines of code.

Destructor

De-allocates the sacred timeline. If you find a branch that was not pruned just `cout` that "Branch was not pruned", but don't attempt to de-allocate it here as all de-allocations should be done by the prune function. Because it is simple de-allocation of a nexus free sacred timeline it should be very short and simple function.

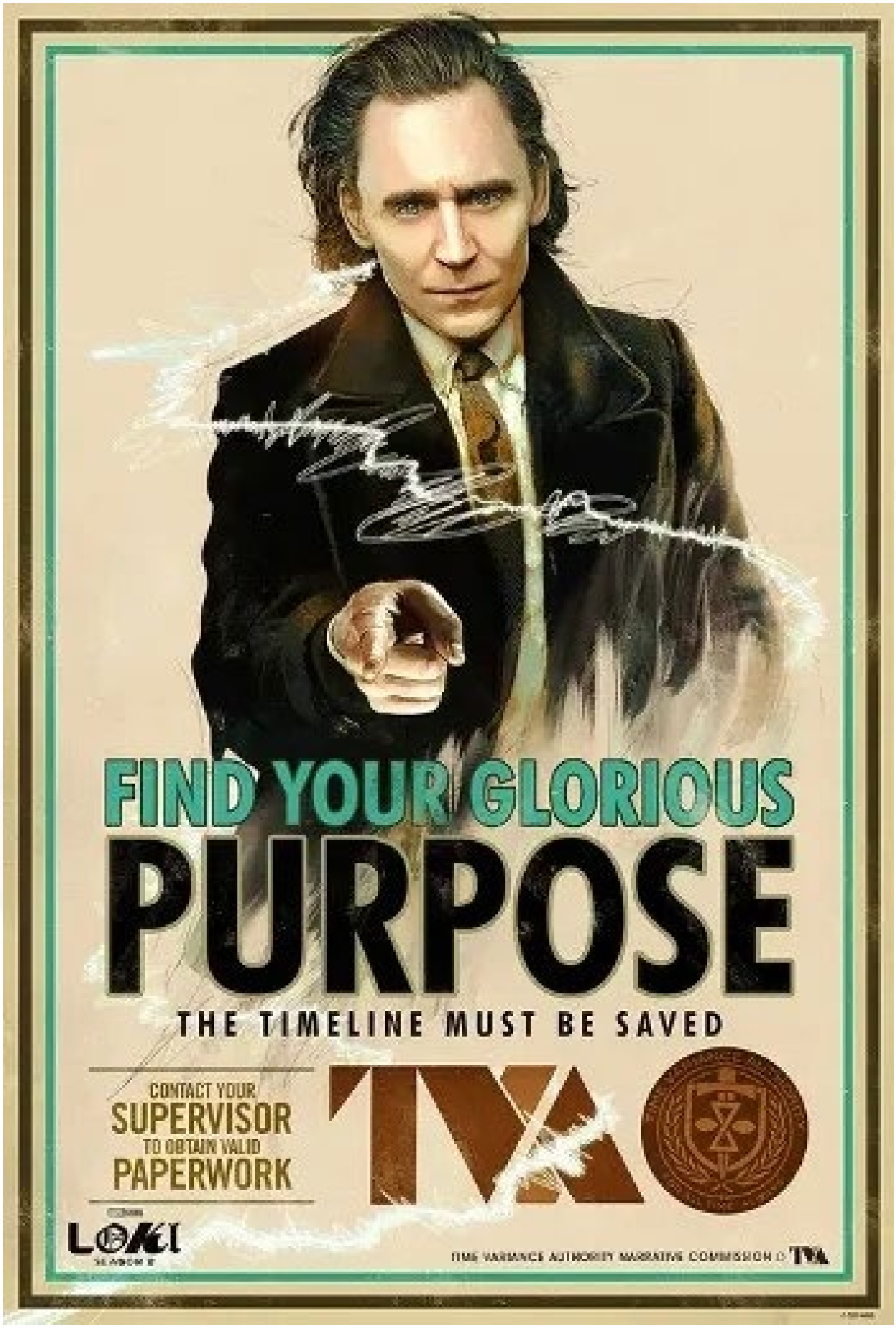
Wtf

This is an obfuscated function that I have given to you to help test and grade your code. Do not touch it and do not call it from anywhere else in your code. The output of this function is identical to what the print function will do. So, you should try to match the output and of course make sure it works with your code.

Overview of main function:

The main function first either uses 4 hardcoded values or reads the ones passed through the command line arguments: **`./a.out 40 4 1970 50`** Let's understand what these values are: 40 is the seed for the random number generator; 4 is number of nexus events to create; 1970 is starting year of the sacred timeline; lastly, 50 is the length of the sacred timeline. These also happen to be the hardcoded values if you were to just run **`./a.out`** but of course when testing your code we will run different values for these. Sample output files have been provided with multiple values to give you ample test cases to ensure your program works. We may also test with additional values we do not share so make sure you thoroughly test your code beyond the provided sample output cases. Each sample output file contains the input values on the first line of it. Also make sure you test with valgrind to ensure you have no memory leaks or errors in your code.

After the command line arguments are read, the seed is used to initialize `srand` and then the timeline is created. Then it is printed out in text then the nexus events are created, then timeline is printed with both the text and visual format then the nexus events are pruned one at a time and the timeline is printed in text. Finally, once the sacred timeline is restored it is printed in text and visual format and lastly the TVA's glorious logo is printed. **For all time. Always.**


FIND YOUR GLORIOUS
PURPOSE
THE TIMELINE MUST BE SAVED
CONTACT YOUR
SUPERVISOR
TO OBTAIN VALID
PAPERWORK
TVA
LOKI
SEASON 2
TIME VARIANCE AUTHORITY MARKETING COMMISSION