Ahmed Khan
Adam Freed
Professor Chenhansa
CS-124
30th March 2023

Lab3 Pre-Lab Report

Purpose:

The purpose of this lab is to be the culmination of everything learned in the Ohlone C++ curriculum, with advanced data structures utilized for real world applications such as encryption and compression, fields which still have room for innovation. Through the self implementation of data structures such has hash tables and trees, an intuition is built for where else they can be utilized. Trees especially are a fundamental element of discrete structures, in logic and mathematics who's algorithms take longer than the age of the universe to solve by brute force. Such discrete mathematics are the foundation of our instant search algorithms that rank trillions of results in fractions of a second, the burgeoning field of AI that is quickly demonstrating aspects of surpassing human capabilities, and encryption upon which we entrust every personal aspect of our lives. This preliminary experimentation serves as a stepping stone to the big leagues of computer science.

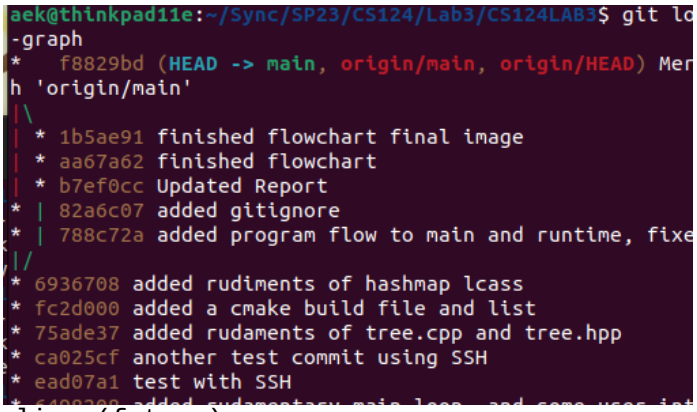
What this program does is allow the user to effortlessly present the computer with data and need not be explicit with instructions, as the system is designed to discern what its objectives must be. The program will decide whether to encrypt given plain text, or vice versa, and adapt to unforeseen instances of foreign characters in the data. The program's utility will be made further robust via an adaptable encryption system, allowing the user to present an alternative cipher like an Enigma Machine and change the encryption format completely at will. This serves as an example of staying one step ahead of ever-present cyber-security threats in the world. Lastly, the program will be memory safe and efficient, employing professional practices during runtime and displaying 0 leaked memory afterwards, meaning it will serve the user and not impede them in any way.

Plan:

Will first implement a runtime class to link with main and provide with main_loop and basic prompts for starting, stopping, and restarting the program. IO will include string input verification as well as file existence verification. Will begin development on a hashmap class, and import the preexisting tree class from another project and adapt it to function as a binary tree. These can happen concurrently since these data structures will be black box implementations that have no dependencies on the program. Will have the crypt class have the functionality to default construct the tree and its respective hash map unless presented with an alternative cipher, where it will then override and propagate them into both data structures. Finally, have a discerner function that can detect whether a file is arriving encrypted or plain text. This comes last as we will need fully functional and perfected encryption and decryption algorithms that we will first hard-code to pass through either channel so we know it works before letting the program decide what to do with data, as we can only judge its functionality based on results we know to be accurate in the first place.

Our encoder will be based on a tree of (character, cipher value) pairs, since it is hard to write a hash function for a character input without relying on its ascii value, but it's easy to compare characters and navigate a binary tree. Furthermore, tree cannot be used for a decoder as the encrypted binary representations of the ascii characters given in the user provided alternative cypher may not conform to a binary search / Huffman tree standard. Our decoder will instead rely on a hash table of (cipher value, character) pairs, because it is easy to write a hash function that translates a series of ones and zeros into an integer. Our current plan for the hash function is to treat the cipher key as a number in binary, convert it into a hash, modulo the size of the array to get the bucket index, as we we plan to use separate chaining for collision management.

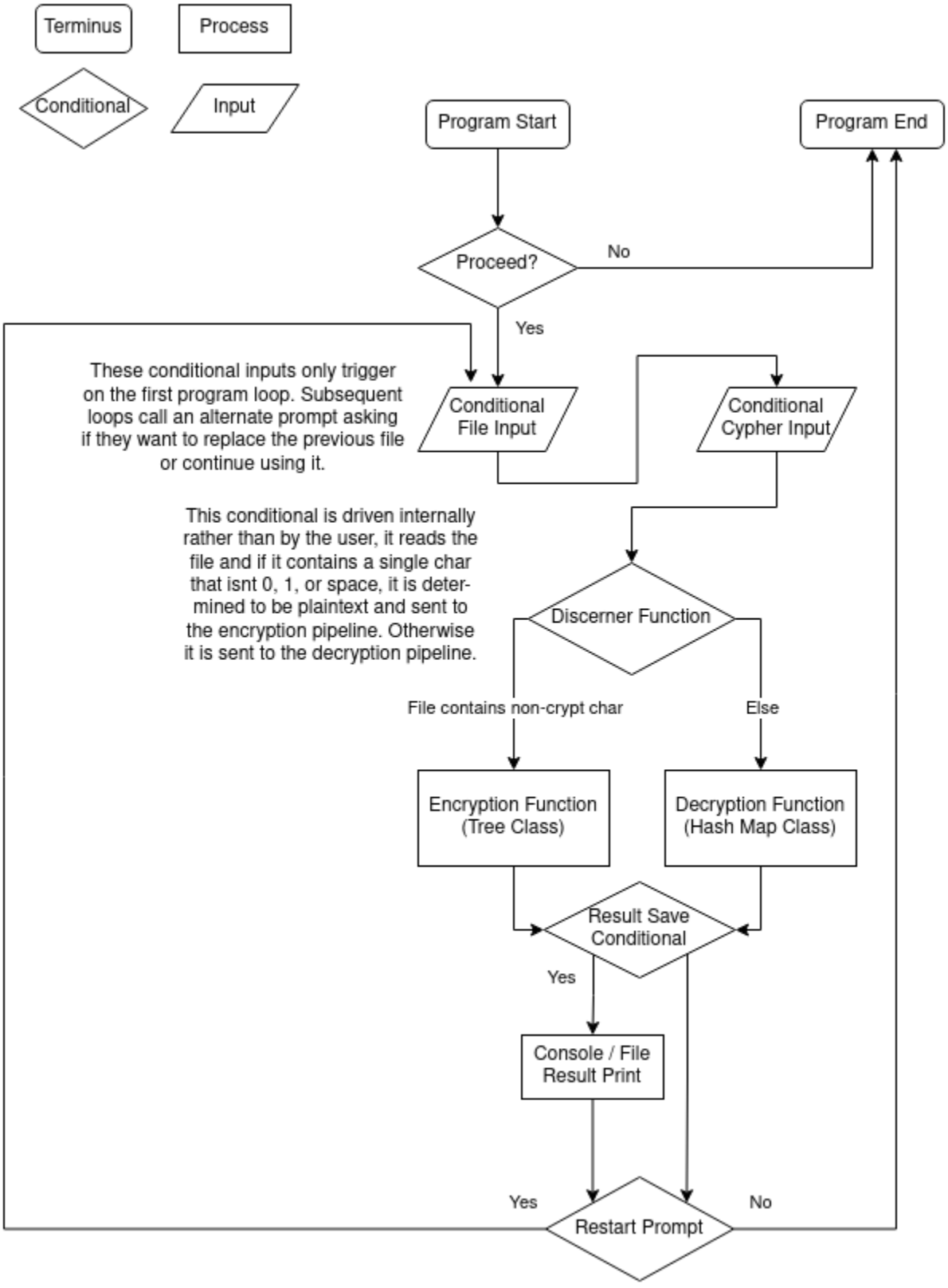
Timeline (past):

- Ahmed has a personal project with a tree class he implemented months prior, with guard clauses preventing ancestors from nesting within their own descendants, or nesting within themselves. Such safety precautions are crucial to prevent any chance for undefined behavior.
- Need to either research or derive the methods for creating an optimal Huffman compression tree either through statistics or algorithms. Debating whether Huffman should ne used at all since it is not a requirement.
- 3/28/23: We have created a github repository where the program files, flow chart xml, and report documents are housed. This should serve as the means of version control and documentation of change through commits in a collaborative setting, and is a great improvement over replit and google docs. Such professional experience should bode well for the future
- 3/30/23: Report and flowchart drafted, laying groundwork for class structure. Classes will most likely include: Tree (for encryption), Hashmap (for decryption), File (for parsing and saving), Crypt (for data processing implementation and objective discernment), and lastly Runtime to house user IO, global data such as file names and file data, domain state scopes and booleans, and the main loop with enter and exit prompts.
- 3/30/23: experienced first merge conflict and resolution, as Adam finalized the report whilst Ahmed finalized the flowchart and began on code. Ahmed's repository was not up to date with origin when Adam pushed his, and attempting to push Ahmed's changes rendered in failure. It took using creating a gitignore for certain cmake files that were changing every time the program was built, as well as calling git fetch to update my own before successful merge, displayed in git's own bash branch visualizer.
- 

Timeline (future):

- 03/31/23: Write hashmap, and tree classes. Also build encoder and decoder from our binary tree and hashmap implementations.
- 04/01/23: Stay on high alert for pranks. Begin to build file parser and ensure that data integrity is being maintained as it is thrown at previously created encoder and decoder classes.
- 04/02/23: Pass strings through crypt functions that either decrypt or encrypt that rely on the previously implemented data structures with their huffman encoding character schemes.
- 04/03/23: Parser function that reads file contents including spaces into the program. Discerner function that decides whether a given file is to be encrypted or decrypted. File saving functionality. Adam unavailable due to a full schedule.
- 04/04/23: Ahmed unavailable, full schedule from 9 am to 9 pm. Adam will debug existing code and finish whatever hasn't been finished.
- 04/05/23: Rehearse demonstration for demo's the following day, record product video.
- 04/06/23: Final tweaks based on instructor feedback. Finalize report.
- 04/07/23: Submission.

Flowchart:


Terminus
Process
Conditional
Input
Program Start
Program End
Proceed?
No
Yes
These conditional inputs only trigger on the first program loop. Subsequent loops call an alternate prompt asking if they want to replace the previous file or continue using it.
Conditional File Input
Conditional Cypher Input
This conditional is driven internally rather than by the user, it reads the file and if it contains a single char that isnt 0, 1, or space, it is determined to be plaintext and sent to the encryption pipeline. Otherwise it is sent to the decryption pipeline.
Discerner Function
File contains non-crypt char
Else
Encryption Function (Tree Class)
Decryption Function (Hash Map Class)
Result Save Conditional
Yes
Console / File Result Print
Yes
No
Restart Prompt