Type one page describing the design and implementation of your parser in terms of PDA terminology and describe the context-free grammar to generate such sequences.

The program consists of 3 files pda.h, pda.cpp, main.cpp and sequence.txt.

In my main file I create a PDA object, and call the readSequence method of this object.

How is the PDA object constructed? The readSequence method creates a new PDA object. This object will we be passing down the other methods as we do our parsing.

The PDA constructor itnitalized the currentstate pointer to property q0, adds ‘z’ to the stack vector and initalized sequence to emty string.

Readsequence then opens up the sequence.txt file and saves the line in the txt file to a variable, which is then initialized to our PDA sequence property. The evaluate loop will be run until the index property of the object is equal or greater than the size of the sequence string. If the code doesn’t exit while running the evaluate function output “Correct sequence” and end the function.

So how does the PDA evaluate work? Takes one argument, which is the PDA object created in the readSequence function. The method ifOrElseSequence will be called determining whether the current element in the sequence is an if or else. If it is an “if” then return 0, otherwise return 1. If it is an “if” stay in q0 and push 0 onto the stack, increment the index by 2. If it is an “else” then check it the top element of the stack is a “z” character. If the top of the stack is a “z” character output , Sequence is syntactically incorrect, and exit the program. It z is not the top of the stack then call the popStack method, which will erase the first element in the stack vector. Add 4 to the index.

The formal defentition of the my pushdownautomata would be M = (Q, Σ, Γ, δ, qs, Z, F).

Where Q: {q0, q1} , Σ: {if,else}, Γ: {z, 0, 1}, qs =q0, Z = z, F = {q0, q1}

The context-free grammar which generate this sequence is

G = ({S, S1}, {if, else} ,S, P)

P: S 🡪 ifS1 | **λ**

S1 🡪S | elseS | **λ**