**Compiler Design Project**

**Introduction**

Compiler design is a fundamental discipline that lies at the heart of modern programming languages, acting as the bridge between human-readable code and efficient machine execution. Compiler design involves transforming high-level programming languages into low-level machine code, encompassing stages such as lexical analysis, syntax parsing, semantic analysis, code optimization, and code generation. With its paramount role in enabling the translation, optimization, and error detection of programs.

**Project idea:**

The idea of our project is to build a simple compiler that checks on the syntax of Declaration, syntax of While loop and syntax of Switch cases.

Also, to make a memory for all variable we have in the code provided.

**The while syntax:**

**A picture containing text, screenshot

Description automatically generated**

**In this code:**

* The code initializes a copy of "Final\_List" as "While\_Syntax\_Check" and sets a counter and flag variable to zero and False, respectively. An "order\_list" is defined to represent the expected order of tokens in a while loop. The code then iterates through each token in "While\_Syntax\_Check" and compares it with the corresponding token in "order\_list." If the tokens match and are in the expected order, the counter is incremented, and the flag is set to True. If any token does not match the expected order, the counter is not incremented. After the loop, if the counter does not match the length of "order\_list" and the flag is True, an error message is appended to the last item in "While\_Syntax\_Check." Finally, the "Final\_List" is updated with the modified "While\_Syntax\_Check," and the "Output" method is called with "While\_Syntax\_Check" as an argument.

**The switch syntax:**

A screen shot of a computer program

Description automatically generated with low confidence

**In this code:**

* various counters and flags required for tracking specific tokens and their occurrences. The "order\_list" is defined to specify the expected order of tokens within a switch statement.
* iterate through each token in "Final\_List" using nested loops. For each token encountered, it compares it with the corresponding token in "order\_list" and increments the counter if there is a match. Additionally, the code checks for specific tokens such as 'switch,' 'case,' ':,' 'break,' and 'default' and increments their respective counters.
* After the loop, the code performs additional checks to ensure the correct structure of the switch statement. It examines the positions of 'default' and 'case' tokens within "Final\_List" to determine if there are any misplaced or missing case statements after the default. If any errors are found, appropriate error messages are appended to the last item in "Final\_List."
* The code checks for conditions where the switch statement structure is not correct. This includes scenarios where the number of 'case' tokens does not match the number of 'break' tokens, where the number of 'case' tokens does not match the number of ':' tokens minus one, or where there is a missing 'switch' statement altogether. In such cases, error messages are appended to the last item in "Final\_List.".
* Finally, the modified "Final\_List" is passed as an argument to the "Output" method, which presumably handles the display.