Zara is a new programming language that has data types, strings, float, integer, arrays and stack. The operators are equal sign, multiplication, division, subtraction, greater than, equal. Furthermore, it has if else, do while, and for loop. Finally, it has sub-programs or methods plus classes.

NB: this work will be done in repLit(create a group and invite me and the tutor)

Week 1: Overview of Compiler and Symbol Table

Assignment:

- Implement a basic symbol table for Zara, supporting variable declarations with their types (integer, float, string, array, and stack). Ensure the symbol table allows adding, updating, and retrieving symbols.
- Write a Zara program with various data types and sub-programs, and verify the symbol table's accuracy by logging symbol entries.

Week 2: Lexical Analysis

Assignment:

- Create a lexical analyzer for Zara that identifies tokens like keywords (`if`, `else`, `do while`), operators, data types, identifiers, and literals. Use regular expressions for token recognition.
- Test the lexical analyzer on sample Zara code to verify correct tokenization.

Week 3: Syntax Analysis - Top Down

Assignment:

- Develop a top-down parser (using recursive descent) for a subset of Zara's grammar, focusing on expressions, control structures (`if-else`, `for`, `do while`), and sub-programs (methods).
- Write Zara programs to test your parser, ensuring it correctly handles valid syntax and flags errors.

Week 4: Syntax Analysis - Bottom Up

Assignment:

- Implement a bottom-up parser (Shift-Reduce or LR) for Zara, covering expression evaluation, conditionals, and loops. Ensure that it can detect conflicts and resolve shift-reduce ambiguities.
- Test the parser by analyzing Zara programs and ensuring correct parsing.

Week 5: Semantic Analysis

Assignment:

- Implement a semantic analyzer that checks for type consistency (e.g., no implicit type conversions), scope rules, and function/method usage. Extend this to handle array and stack type consistency.
- Test the analyzer with Zara programs, demonstrating correct and incorrect usages of types and scope.

Week 6: Syntax-Directed Translation

Assignment:

- Implement syntax-directed translation rules for Zara to translate high-level constructs (such as expressions, loops, and sub-programs) into intermediate representations.
- Use synthesized attributes to generate intermediate code as you parse Zara programs, ensuring it works for a variety of constructs.

Week 7: Intermediate Representation

Assignment:

- Generate intermediate code for Zara programs using a format like three-address code (TAC). Focus on expressions, control structures, and method calls.
- Test your implementation with Zara code samples and verify the generated intermediate code.

Week 8: Code Optimization

Assignment:

- Implement code optimization techniques such as constant folding, dead code elimination, and loop invariant code motion on the intermediate code generated in Week 7.
- Write Zara programs with potential optimizations and show the improvements after applying the optimization techniques.

Week 9: Object-Oriented Code

Assignment:

- Extend the Zara compiler to support object-oriented features like classes, inheritance, and methods. Implement class hierarchies with member functions and demonstrate method overloading or inheritance.
- Test object-oriented constructs in Zara programs, ensuring the compiler handles object creation and method calls correctly.

Week 10: Machine Code

Assignment:

- Extend the Zara compiler to generate low-level assembly or machine code for Zara constructs such as arithmetic expressions, control flow, method calls, and object creation.
- Write Zara programs and show the corresponding machine code generation.

Week 11: Error Handling and Recovery

Assignment:

- Implement robust error detection and recovery mechanisms for the Zara compiler. Handle lexical, syntactic, and semantic errors, providing detailed error messages and recovery options where appropriate.
- Test your compiler with Zara programs containing errors and demonstrate how it handles and recovers from those errors.