|  |  |
| --- | --- |
| **https://lh7-us.googleusercontent.com/cqr1zywKLZ-KYtGHQsJs_4r0Pz65g7Hm9cYAB_QVFrqm6JK4FqjMzqYGew6RHuFzmfT56Wdn2C69ISfHmuDwuy_tnadpQXO2ujqRH_tBzkPaOHrK6awj4voQaDIwnQBQeNfh8u1bi15aREV24NpZmw** | **Compiler Construction**  **BSCS 5-A**  **Department of Computer Science**  **Bahria University, Lahore Campus** |

**Assignment: [3 & 4]**

Date: Week 2, March 2024

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Roll No:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| **Evaluation of CLO** | **Question Number** | **Marks** | **Obtained Marks** |
| **CLO3:  Design and implement a compiler using a software engineering approach**  **CLO4: Select and employ appropriate code generation and optimization techniques** | 1 | 10 |  |
|  |
| **Total Marks** | | 10 |  |

**Q1: Implementing a Compiler for an Object-Oriented Programming Language with Fibonacci Sequence Generation**

**Create a simple object-oriented programming language, which we'll call "OOPL." OOPL should support the following features:**

1. Variables of type integer, float, and string
2. Basic arithmetic operations (+, -, \*, /) for integer and float types
3. String concatenation using the "+" operator
4. Conditional statements (if-else) with boolean expressions
5. Loops (while) with conditional expressions
6. Functions with parameters and return types
7. Classes with attributes and methods
8. Inheritance and polymorphism
9. A built-in function to generate the Fibonacci sequence up to a given number

**Task**

1. The OOPL language should have a built-in function fib(n) that generates the Fibonacci sequence up to the n-th number. For example, fib(5) should return the sequence 0, 1, 1, 2, 3.
2. Design a non-deterministic finite automaton (NFA) and a deterministic finite automaton (DFA) that can recognize the Fibonacci sequence generated by the fib(n) function. The NFA and DFA should be able to recognize the sequence of numbers in the Fibonacci sequence, regardless of the input n.
3. Implement a parsing technique suitable for compiling OOPL programs. You can choose from top-down parsing (e.g., recursive descent parsing) or bottom-up parsing (e.g., LR parsing). Justify your choice of parsing technique and explain how it handles the syntax of OOPL.

**Constraints**

1. The NFA and DFA should have a finite number of states (less than 20).
2. The parsing technique should be able to handle left recursion in the OOPL grammar.
3. The compiler should be able to generate assembly code for the target machine that is efficient in terms of execution time and memory usage.

**Deliverables**

1. A written report detailing the design and implementation of the compiler, including the lexical analyzer, parser, semantic analyzer, code generator, and optimizer.
2. The source code for the compiler, including any necessary libraries or tools.
3. A test suite of OOPL programs that demonstrate the functionality of the compiler, including programs that use the fib(n) function.
4. A detailed description of the NFA and DFA designs, including state transition diagrams and explanations of how they recognize the Fibonacci sequence.
5. A justification of the chosen parsing technique, including an explanation of how it handles the syntax of OOPL.