Date of Midterm ___________________

Please save this study guide. It is part 1 of the study guide for the final.

Basic Topics

- *Chapter 1*: Introduction
  - Organization of a compiler
    - Lexical Analyzer (Scanner)
    - Syntactical Analyzer (Parser)
    - Semantical Analyzer (Checker)
    - Optimizer
    - Code Generator
      - Cross Compiler
      - Native Code Compiler
  - Automated Tools
    - Lex
    - Yacc
  - Basic data structures of a compiler
    - Tokens
    - Symbols Table
    - Abstract Syntax Tree
- *Chapter 2*: Scanning
  - Regular Expression
  - Extensions to Regular Expressions
  - Deterministic Finite Automaton
  - Non-Deterministic Finite Automaton
  - Thompson's Construction
  - Subset Construction
- *Chapter 3*: Context-Free Grammars and Parsing
  - Terminals and Non-Terminals
  - Productions
  - Derivations: Left-Most and Right-Most
  - BNF and EBNF
  - Parse Tree and Ambiguous Grammars
  - Precedence and Associativity
  - Disambiguating Rule
- *Chapter 4*: Top-Down Parsing
  - Removing Left Recursion
  - Left Factoring
  - Predictive Parsing
    - Recursive Descent
    - LL(1)
  - First and Follow sets

1. What are the basic parts of a compiler? Which parts depend on the source language, and which parts are independent of the source language?

2. What are the three major data structures used by a compiler?

3. What is the definition of a regular expression?

4. What is a **cross-compiler**? (One that is mad at you for making a syntax error??)

5. Suppose a number in a programming language is defined as follows. There is an optional + or – sign, there must be at least one digit followed by a decimal point, and at least one digit on the right of the decimal point. Thus 0.0 is the minimum way to expressing the value zero.

    a. Write a **regular expression** for such a number.
    b. Create a **DFA** that accepts such numbers.
    c. Use **Thompson's Construction** to create a NFA to accept these numbers.
    d. Use the **Subset Construction** to convert the NFA in (c) to a DFA.
    e. Did you get the same DFA in (d) as you had in (b)?

6. Using **pseudocode**, outline a method that would implement the DFA from 5(b). The level of detail for the pseudocode should be similar to that of figure 2.7 on page 62.

7. In the subset construction, what is meant by an **epsilon-closure** of a state? How would you describe the epsilon closure of a set of states?

8. There is a software tool, written by Mike Lesk, that will generate automatically generate the code for a scanner. What is this program called?

9. What is meant by the term **formal language**?

10. In the *Chomsky Hierarchy*, there are regular grammars (type 3 grammars), context-free grammars (type 2 grammars), context-sensitive grammars (type 1 grammars), and phrase-structured grammars (type 0 grammars). Which one of these is the type of grammar for almost all programming languages?

11. What does it mean when a grammar is **ambiguous**? Can an ambiguous grammar be used for a programming language? Explain your answer.

12. Given the following grammar

```
Exp → Exp Addop Term | Term
Addop → + | -
Term → Term Multop Factor | Factor
Multop → *
Factor → ( Exp ) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9
```

a. Show a **left-most** derivation for 2 * (3 + 4)
b. Draw a **parse tree** for this derivation.
c. The rule for `Exp` is **left-recursive**. How could this rule be modified to remove the left-recursion?

13. What do the acronyms **BNF** and **EBNF** represent?

14. In a context-free grammar, what is the difference between a **terminal** symbol and a **non-terminal** symbol? When a symbol is called **nullable**, what does it imply? Can both terminals and non-terminals be nullable?

15. One software tool, written by Steve Johnson, is called Yacc. For what does the acronym **Yacc** stand? What is the output of this tool?

16. If I have a grammar rule `X → X OP X | Y`, what is the **associativity** of OP?

17. What is the purpose of the **parser** component of a compiler? What is the **input** to the parser component? What is the **output** of the parser component?

18. A **predictive** parser falls into the category of a top-down parser. Why is this?

19. Given the following parsing algorithms: recursive-descent, LL(1), LR(1), SLR, and LALR, which ones are top-down algorithms?

20. When a parser is written by hand, not produced by an automated tool, what parsing algorithm is most often used?

21. Given the grammar rule `X → Y Z | Y`, this rule causes problems for a top-down parser. Why is this? To use this rule with a top-down parser, it must be **left-factor**. What would this rule look like if left-factored?

22. An LL(1) parser is usually **table-driven**. Describe the organization of the table used by an LL(1) parser.