Midterm. (200 Points)

Conditions: You are permitted writing utensils, the midterm, one crib-sheet. No other items are permitted.

Note: The length of this practice exam does not necessarily represent the length of the midterm (in terms of time needed or number of questions).

Topics:

Coverage: All material covered in class and corresponding Book Chapters. Particularly focus on the following topics. You will be allowed 1 crib sheet double-sided, hand-written. I provide algorithms in this practice for example, the **algorithms will NOT be provided in the exam** and thus you should include pertinent algorithms and other important information on your crib sheet.

On your crib sheet, include (at the least) algorithms for building an LL(1) parse table, performing an LL(1) parse using a table, performing an LR(1) parse using a LR(1) table. You should be able to show all stages of the parse using a table.

I. Compiler and Interpreter Design
II. Context Free Grammars
  II.1 Operator precedence
  II.2 Operator associativity
  II.3 BNF Productions
III. Regular Languages and Lexers
  III.1 Implementations of Lexers
  III.2 Regular Expressions: Be able to design an regex and convert a regex to NFA
  III.3 DFA and NFA: be able to create DFA from NFA
IV. Top Down Parsing
  IV.1 Recursive implementations
  IV.2 Table implementations
  IV.3 Know FIRST and FOLLOW
  IV.4 Disjointness Test
  IV.5 Left Factoring
  IV.6 Limitations
  IV.7 Be able to convert a non-LL(1) grammer into an LL(1) grammar (when possible)
V. Bottom Up Parsing
  V.1 Limitations
  V.2 Handles
  V.3 Table implementations

1. Compilers vs Interpreters

Multiple choice. (circle one only)

1.1 Translation is performed in a _____

A. Compiler
B. Interpreter
C. Both
D. Neither

1.2 Programming languages implemented as a(n) _____ generally provide for a faster execution time.

A. Compiler
B. Interpreter
C. Both
D. Neither

Application and Concepts.

1.3 Discuss and analyze the Readability and Writability of C++ using the following snippet of C++ code. Using examples in the code below, identify one operator or structure that provides for increased readability. Identify one operator or that structure that decreases readability, but benefits writability. Explain.

```
for( int i = 0; i < n; i++ ){
        cin >> val;
        sumTotal += val;

}
```

2 Regular Grammars + Lexical Analysis

Multiple Choice. (circle only one)

2.1 A regular language is recognized by a _____

A) finite state automaton
B) regular expression
C) token
D) lexeme

2.2 A regular language is generated by a _____

A) regular expression
B) finite state machine
C) BNF productions
D) tokens

Short Answer.

2.3 Describe the token that is lexed by the following pseudocode:

```
// pointer pos points to next char in input buffer

void lex_X(){
char ch = getChar(); // pulls char from input buffer
if ( isDigit(ch) )
        {addChar();  pos++;}
         // if is numeric digit, add to current lexeme
ch = getChar();
pos++;
while( isDigit(ch)  ){
        addChar();
        pos++;
        ch = getChar();
}
```

3 Context-Free Grammars

Multiple Choice. (circle only one)

3.1 A context-free language is generally defined by a _____ .

A) a recognizer
B) a generator
C) either a recognizer or a generator
D) parser

3.2 Given the grammar below. What is the order of evaluation (order of precedence from high to low) of the binary operators for input sentence: 3+4^2+5 ? Consider operator precedence and associativity.

<e> => <e> * <t> | <t>
<t> => <v> + <t> | <z>
<z> => <v> ^ <z> | <v>
<v> => x | y | z | 0 … 9

A. ^, + (left), +(right)
B. ^, + (right), +(left)
C. + (left), +(right), ^
D. + (left), ^, +(right)

Concepts.

3.3 A language can be formally defined as a set of sentences. A context-free grammar is a set of rules that determine and recognize whether a sentence belongs to a particular language or not. Given this definition it is easy to see that a grammar, G, will imply a language, L, ie, G => L. In other words, a grammar will fully determine a language. Is the converse true? Does a language determine the grammar. That is, if we have a language can we determine the grammar? In other words, does L => G?

Application.

3.4 Using the following Grammar, derive the sentence: x = x+++1 , using a rightmost derivation. If a derivation does not exist for the sentence, state why.

<assign> => <var> = <expr>
<expr> => <expr> + <expr> | <incr> | <var>
<incr> => <var>++ | <num>
<var> => x
<num> => 1

5 Top-Down Parsing

Multiple Choice. (circle only one)

5.1 The job of a parser is to _____.

A) Build a parse tree
B) Accept or Reject an input
C) Both A and B
D) Neither A, B nor C

Concepts / Short Answer.

5.2 Name 2 limitations of top-down parsers.
Left recursion

5.3 Can the following grammar be parsed with an LL(1) parser? Why or why not?
A => bA | aA | CB
B => bbB | ε
C => c | d

Application.

5.4 Using the LL parse table and the LL parse algorithm (all below), parse the sentence x+y+x . Fill in the stack trace table below to show the steps of the parse (add more rows as needed). State whether the input is accepted or not.

	x	y	+	$
<e>	<e> => <t><e'>	<e> => <t><e'>		
<e'>			<e'>=> + <t><e'>	<e'>=> ε
<t>	<t> => x	<t> => y		

Stack	Input

```
set pos to first symbol
set stack to S$ // S = startSymbol
Let: X = peek(); a = input [ pos ] ;
do {
        if X is terminal or $ {
                if X == a
                        {pop X; pos++; }
                else
                        syntax error
        }
        else if M[ X, a] ==  X=>α1 , … αk {
                pop X;
                push αk , … α1 ;}
        else error
} while( X != $ )
// accept
```

5.5 Using the LL parse table and the LL parse algorithm (all below), parse the sentence int*(int) . Fill in the stack trace table below to show the steps of the parse (add more rows as needed). State whether the input is accepted or not.

	int	+	*	(	)	$
E	E=>TE'			E=>TE'		
E'		E'=>+TE'			E'=>ε	E'=>ε
T	T=>FT'			T=>FT'		
T'		T'=>ε	T'=>*FT'		T'=>ε	T'=>ε
F	F=>int			F=>(E)		

Stack	Input

```
set pos to first symbol
set stack to S$ // S = startSymbol
Let: X = peek(); a = input [ pos ] ;
do {
        if X is terminal or $ {
                if X == a
                        {pop X; pos++; }
                else
                        syntax error
        }
        else if M[ X, a] ==  X=>α1 , … αk {
                pop X;
                push αk , … α1 ;}
        else error
} while( X != $ )
// accept
```

6 Bottom-Up Parsing

Short Answer.

6.1 Shift Reduce parsers build a parse tree that represents a _____ derivation .

A) leftmost
B) rightmost
C) topmost
D) bottommost

Concepts.

6.2 Shift: explain the shift operation (in terms of constructing the parsing tree) in a shift reduce parser.

6.3 Reduce: explain the Reduce operation (in terms of constructing the parsing tree) in a shift reduce parser.

6.4 The stack trace below shows the state of the stack and input buffer during a bottom up parse. This partial derivation (a derivation that is not complete) has a corresponding partial parse tree (a parse tree that is not complete). Draw the corresponding partial parse tree for the last row shown in this stack trace.

Stack	Input	Action
0	id * id$	Shift 5
0 id 5	* id$	Reduce 6
0 F 3	* id$	Reduce 4
0 T 2	* id$	Shift 7

Application.

6.5 Using the SLR parse table and the LR parse algorithm (all below), parse the sentence id + id . Fill in the stack trace table below to show the steps of the parse (add more rows as needed). State whether the input is accepted or not.

```
•     // Assume we have an LR parse table
push startState = 0;
do{
        s = peek(); a = input[ pos ];
        if action[ s, a ] == shift s'{
                push a; push s'; pos++;
                }
        else if action[ s, a ] == reduce A => α{
                pop 2*| α|;
                s' = peek();
                push A; push goto[ s', A ];
                // chosen handle A => α
                }
        else if action[ s, a ] == accept
                return accept;
        else
                return error;
}while true
```

Stack	Input

	Action						Goto		
State	id	+	*	(	)	$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Rules

1. E => E + T
2. E => T
3. T => T * F
4. T => F
5. F => (E)
6. F => id