EE573 Midterm Exam, Fall 99 **Name** _____

1. (3Pts) Compilers typically translate source programs into assembly code. Give three examples of other compiler input and output forms.

   1.

   2.

   3.

2. (3Pts) A customer wants you to create a tool that can translate a textual input into a well-defined output form. To do this you consider writing either an interpreter or a compiler. What questions do you ask the customer in order to decide between the two methods? Explain your choice.

3. (3Pts) Name three advantages and three disadvantages of a 1-pass compiler compared to a multi-pass compiler.

   Advantage 1

   Advantage 2

   Advantage 3

   Disadvantage 1

   Disadvantage 2

   Disadvantage 3

4. (3Pts)
   (a) What parts of a compiler can be designed with the help of automatic tools?

   (b) What makes it intrinsically difficult for the other parts to be generated automatically?

5. (3Pts) In the statement `a = b + c * d,` explain what is

   (a) the syntax

   (b) the static semantics

   (c) the execution semantics.

6. (6Pts) Write a grammar for the expression a ∩ b ∪ c ∩ d such that the parser follows the precedence ∩ before ∪. Allow parentheses to change the precedence.

7. (8Pts)
   (a) Write a scanner translation table that recognizes the regular expression // ((/|λ) Not(/))* //
   You can use the notation ch1..ch2 and Not(ch) for column headings.

   (b) Add "actions" to the table such that the "//" delimiters get discarded. Allowable actions are "t" for toss and "a(c)" for append character c. You can use the character "cic" to refer to the current input character.

8. (3Pts) Draw a finite automaton for the regular expression (A B)+ ,
   given the automata A: –|O A O|– and B: –|O B O|–

9. (3Pts) The automaton created in (8) is not necessarily optimal. Explain, in general terms, what steps

10. (6Pts) Is the following grammar in standard or extended BNF notation? Turn it into the other form.
    A → B X
    X → , B X
    X → λ
    B → u v w
    B → m n

11. (3Pts) Mark all that is true in the above grammar?
    ( ) A => B
    ( ) A =>$^{+}$ B
    ( ) A =>* B

12. (3Pts) Explain what it means to "predict a production"

13. (6Pts) Describe, in general terms, the steps necessary to generate a parser, given a parse table,

    (a) a recursive descent parser driver

    (b) a stack-based parser driver

14. (3Pts)
    (a) Is the following an LL1 grammar ?

    X → A B C
    A → c d e
    B → c d f
    B → c d g
    C → c d h

    (b) If your answer was yes, add a small modification that makes it non-LL1.
    If no, modify it so it is LL1.

15. (3Pts) Name four different attributes that a symbol table entry may have.
    (1)
    (2)
    (3)
    (4)

16. (9Pts)
(a) Write the assembly code for the program fragment:

```
REPEAT
  <stmts>
  count --
UNTIL count = 0
```

(b) The Motorola 68000 processor had an instruction `SBNE reg,label`, which subtracts 1 from `reg`, then jumps to `label` if the result is not zero. Specify a peephole optimization that takes advantage of the SBNE instruction.

17. (9Pts) Draw the stack frame of the following call to subroutine SX. (Parameters with a VAR attribute are passed by reference, others are passed by value. Each variable and address consumes 1 storage unit. The language does not support nested scopes of identifiers).

```
Reslt = SX(x, 3, y)
```

```
int subroutine SX(int a, int b, VAR int c) {
  // local variables:
  int i1,i2,i3;
  // subroutine code:
  ...
}
```

Give a brief description for each stack frame element (at most 1 line).

18. (6Pts) Mark the common subexpression that can be reused in each statement. In each case, circle and connect the two involved subexpressions. (The expressions are evaluated from left to right.)

```
A = A + B + C
B = A + B + C
C = A + B + C
D = A + B + C
```

19. (6Pts) Draw the expression tree for the expression (a+b)*(c+3)+(a+e)/(f-g)
and annotate the register use for the tree code generation algorithm (same assumptions as in class)

20. (6Pts) Bonus points: The register allocation algorithms discussed in class so far require a modified variable to be written back to memory at the end of each basic block. In reality this operation may not be necessary. What issues do you think will come up when trying to avoid such unnecessary store operations?