Concepts of Programming Languages, CSCI 305, Fall 2020
Exam 4, Nov. 20

Name __Answer Key_____

This is a closed book exam. You may not use notes, the text book, Internet, etc.

1. The static data in a C program is given a layout. At what time did that data get bound to a specific layout? (4 pts.)
    a. Language design time
    b. Compile time
    c. Link time
    d. Load time
    e. Run time

2. The total amount of space occupied by a program code and data is most likely to be bound at what time? (4 pts.)
    a. Language implementation time
    b. Compile time
    c. Link time
    d. Load time
    e. Run time

3. In C the data type integer is allowed to have a certain range of values. At what time are integers bound to this range? (4 pts.)
    a. Language design time
    b. Language implementation time
    c. Compile time
    d. Load time
    e. Run time

4. In what part of memory are scalar variables most likely to be stored when a program is executing? (4 pts.)
    a. Static storage
    b. Stack storage
    c. Heap storage
    d. All of the above
    e. None of the above

5. In what part of memory are machine instructions most likely to be stored when a program is executing? (4 pts.)
    a. Static storage
    b. Stack storage
    c. Heap storage
    d. Any of the above
    e. None of the above

6. Consider the following pseudo code

```
x : integer   -- global

procedure set_x(n : integer)
        x:=n

procedure print_x()
        write_integer(x)

procedure first()
        set_x(1)
        print_x()

procedure second()
        x : integer
        set_x(2)
        print_x()

set_x(0)
first()
print_x()
second()
print_x()
```

a. What does this program print if the language uses static scoping? (3 pts.)

1 1 2 2

b. What does it print if the language uses dynamic scoping? (2 pts.)

1 1 2 1

7. You have been hired by a company which develops air traffic control systems. Due to the real-time and highly complex nature of these systems, they have decided to create an in-house, specialized programming language, to be used in the development of all of their systems. This language is meant to be highly modular and allows nested subroutines. Your team is in the process of deciding whether the language uses static or dynamic scope. They have asked you to describe the difference. Write your response. (3 pts.)

The scope determines where definitions of referenced non-static variables will be found. In the case of static or dynamic scope, the first place to look is within the local procedure. If the variable definition is not there, static scope will look for the declaration in the procedure in which this procedure is declared; whereas dynamic scope will look in the procedure which called the current procedure.

Your response was so helpful and clear that they now want you to make a recommendation of which scope to use and why. Write your response. (2 pts.)

Since air traffic control systems need to be very safe, I recommend using static scope. Systems which use static scope are easier to debug and therefore safer. Dynamic scope gives flexibility, and might be easier to implement. However, flexibility is not as important as safety for air traffic control systems.

8. The following grammar parses strings in the language $\{a^n b^m c^k \mid n, m, k \geq 0\}$. Example strings in this language are ε, accc, b, aaaabcc, aaabbbccc.

abcString → alist blist clist $
alist → a alist
alist → ε
blist → b blist
blist → ε
clist → c clist
clist → ε

Create an attribute grammar for this grammar which gives the length of the string to the root. (5 pts.)

abcString → alist blist clist $
abcString.len = alist.len + blist.len + clist.len

$alist_1$ → a $alist_2$
$alist_1$.len = alist.$_2$len + 1

alist → ε
alist.len = 0

$blist_1$→ b $blist_2$
$blist_1$.len = blist.$_2$len + 1

blist → ε
blist.len = 0

$clist_1$ → c $clist_2$
$clist_1$.len = clist.$_2$len + 1

clist → ε
clist.len = 0

9. Write a BNF or regular expression description for financial quantities which begin with dollar sign ($), contain a number, decimal point (.) and two decimal digits.

   The number must be at least one character but has no maximum number of characters. The number contains thousand, million, billion, etc. dividers. The number can be 0, but if it is not 0, it cannot contain leading 0s.

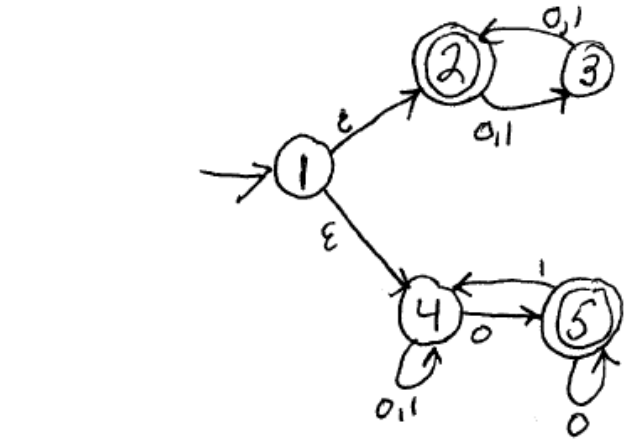
   Negative financial quantifies are not allowed and the decimal portion is required.

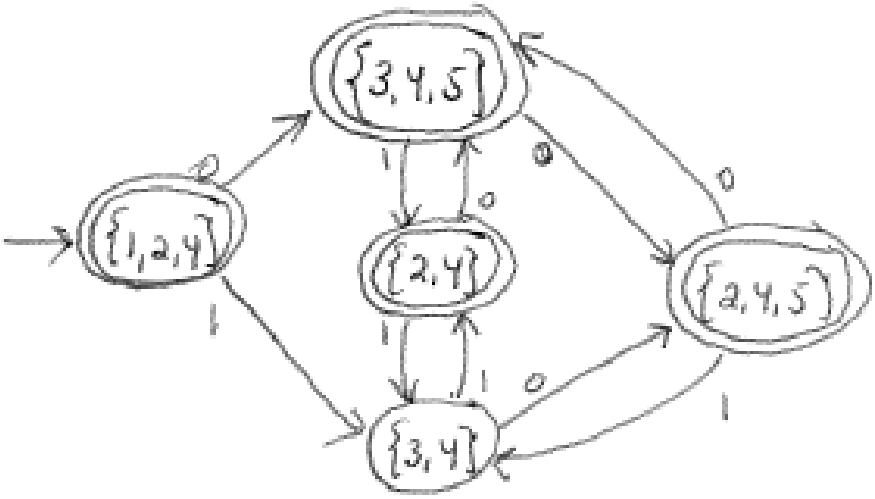
   (10 pts.)

Legal Financial Quantities	**Illegal Financial Quantifies**
$0.00	0.00
$0.59	$139
$2,546,000.99	$04.99
$99.01	$.75

financialQuatity → $ dollars fraction
dollars → 0 | nonzeroD cluster* | nonzeroD digit cluster* |
nonzeroD digit digit cluster *
nonzeroD → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
digit → 0 | nonzeroD
cluster → , digit digit digit
fraction → . digit digit

10. Use the construction described in class and the text to convert the following nondeterministic finite automaton (NFA) to an equivalent deterministic finite automaton (DFA). (4 pts.)





Describe the strings which are accepted by the NFA above. (1 pt.)

Strings from the alphabet {0,1} which are either of even length or end with a 0.

11. Extend the grammar below to include if statements and while loops, along the lines suggested by the following examples:

```
abs := n
if n < 0 then abs := 0 – abs fi

sum := 0
read count
while count > 0 do
        read n
        sum := sum + n
        count := count -1
od
write sum
```

Your grammar should support the six standard comparison operations (<, <=, >, >= , ==, !=) in conditions, with arbitrary expressions as operands. It also should allow an arbitrary number of statements in the body of an 'if' statement or 'while' statement.

(10 pts)

1. program → stmt_list $$
2. stmt_list → stmt stmt_list
3. stmt_list → stmt
4. stmt → id := expr
5. stmt → read id
6. stmt → write expr
7. expr → term
8. expr → expr add_op term
9. term → factor
10. term → term mult_op factor
11. factor → (expr)
12. factor → id
13. factor → number
14. add_op → +
15. add_op → -
16. mult_op → *
17. mult_op → /

Sample answer:

1. program → stmt_list $$
2. stmt_list → stmt stmt_list
3. stmt_list → stmt
4. stmt → id := expr
5. stmt → read id
6. stmt → write expr

   stmt → if cond then stmt_list – stmt_list fi

   stmt → while cond do stmt_list od

   cond → expr comparator expr

   comparator → < | <= | > | >= | == | !=
7. expr → term
8. expr → expr add_op term
9. term → factor
10. term → term mult_op factor
11. factor → (expr)
12. factor → id
13. factor → number
14. add_op → +
15. add_op → -
16. mult_op → *
17. mult_op → /

12. The following grammar has a single variable S and two terminals 0 and 1.

S → 0 S 1
S → 01

Tell if this grammar is LL(1), and if not, convert it into an LL(1) grammar. (5 pts)

This grammar is not LL because it has a common prefix.
The following is an equivalent LL grammar

S → 0 S_tail1
S_tail → 1 | S 1

13. Define what is meant by orthogonality in the context of programming languages. (5 pts.)

Orthogonality is when features can be used in any combination and the meaning of a feature is consistent, regardless of the other features with which it is combined.

14. The grammar

S → E $
E → E + T | T
T→ T * F | F
F→ (E) | id

can be adapted for top-down parsing as follows:

1. S → E $
2. E → T E'
3. E'→ + T E' | ε
4. E'→ ε
5. T→ F T'
6. T'→ * F T' | ε
7. T'→ ε
8. F→ (E) | id
9. F→ id

This adapted grammar has the LL parsing table:

	+	*****	**(**	**)**	**id**	**$**
S	-	-	1	-	1	-
E	-	-	2	-	2	-
T	-	-	5	-	5	-
E'	3	-	-	4	-	4
F	-	-	8	-	9	-
T'	7	6	-	7	-	7

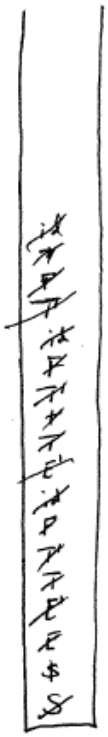
```
terminal = 1 . . number_of_terminals
non_terminal = number_of_terminals + 1 . . number_of_symbols
symbol = 1 . . number_of_symbols
production = 1 . . number_of_productions

parse_tab : array [non_terminal, terminal] of record
      action : (predict, error)
      prod : production
prod_tab : array [production] of list of symbol
-- these two tables are created by a parser generator tool

parse_stack : stack of symbol

parse_stack.push(start_symbol)
loop
      expected_sym : symbol := parse_stack.pop()
      if expected_sym ∈ terminal
            match(expected_sym)                    -- as in Figure 2.17
            if expected_sym = $$ then return       -- success!
      else
            if parse_tab[expected_sym, input_token].action = error
                  parse_error
            else
                  prediction : production := parse_tab[expected_sym, input_token].prod
                  foreach sym : symbol in reverse prod_tab[prediction]
                        parse_stack.push(sym)
```

Using the above LL parsing code, the parsing table and productions given previously, walk through the parse of the string

a + a * b $

clearly showing every item that is push and popped from the parsing stack.

(10 pts.)



a+a*b$

table (S, id) gives 1
table (E, id) gives 2
table (T, id) " 5
table (F, id) " 9
match ids
a+a*b$
table (T', +) " 7
table (E', +) " 3
match +s
a*a*b$
table (T, id) " 5
table (F, id) " 9
match ids
a*a*b$
table (T', *) " 6
match *s
a*a*b$
table (F, id) " 9
match ids'
a+a*b$
table (T', $) " 7
table (E', $) " 4
match $s
a*a*b$

3

14. Complete the grid to contrast the imperative, functional and logic languages. (10 pts.)

	Imperative	**Functional**	**Logic**
Basis	Turing machines (Alan Turing)	Lambda calculus (Alonzo Church)	Mathematical logic (Aristotle)
Computes principally using	Iteration and side effects	Substitution of parameters into functions	Resolution of logical statements, driven by the ability to unify variables and terms

Write code in Prolog for the following. Do not use the computer, simply write your program onto the test page.

15. Write Prolog code to reverse a list. So that

```
?- my_reverse([a, b, c, d],X).
X = [d, c, b, a].
```

(5 pts.)

SWI Prolog contains a reverse function, however, do not use this in your answer.

Might be helpful: The Prolog relation append is a set of tuples of the form (X,Y, Z), where Z consists of the elements of X followed by the elements of. Y.

Example: ([a],[b],[a, b]) is in the relation append.

```
my_reverse([],[]).
my_reverse([H|T],Ans):- my_reverse(T,SubAns),
                        append(SubAns,[H], Ans).
```

16. In the Prolog version of the Tic-Tac-Toe game, given in lab and the text, the user is ‘o’ and the computer is ‘x’. The AI for selecting the computer’s move follows:

```
% Strategy
good(A) :- win(A).
good(A) :- block_win(A).
good(A) :- split(A).
good(A) :- strong_build(A).
good(A) :- weak_build(A).
good(5).
good(1).
good(3).
good(7).
good(9).
good(2).
good(4).
good(6).
good(8).

win(A) :- x(B), x(C), line(A,B,C).
block_win(A) :- o(B), o(C), line(A,B,C).
split(A) :- x(B), x(C), different(B, C),
                    line(A,B,D), line(A,C,E), empty(D), empty(E).
strong_build(A) :- x(B), line(A,B,C), empty(C), \+(risky(C)).
weak_build(A) :- x(B), line(A,B,C), empty(C), \+(double_risky(C)).
risky(C) :- o(D), line(C,D,E), empty(E).
double_risky(C) :- o(D), o(E), different(D, E),
                    line(C,D,F), line(C,E,G), empty(F), empty(G).
```

Say that you have chosen to first. Give a sequence of moves that eventually cause the rules

```
good(A) :- win(A).
good(A) :- block_win(A).
good(A) :- split(A).
```

to fail, but the rule

```
good(A) :- strong_build(A).
```

to succeed. (3 pts.)

There are many possible answers:
User chooses 2, which causes the computer to choose 5. User chooses 7, which causes the computer to choose 4 in a strong build.

Alternatively,
User chooses 2, which causes the computer to choose 5. User chooses 9, which causes the computer to choose 6 in a strong build.

Alternatively,
User chooses 4, which causes the computer to choose 5. User chooses 2, which causes the computer to choose 1 in a strong build.

Alternatively,
User chooses 4, which causes the computer to choose 5. User chooses 6, which causes the computer to choose 8 in a strong build.

Alternatively,

User chooses 6, which causes the computer to choose 5. User chooses 2, which causes the computer to choose 3 in a strong build.
Alternatively,
User chooses 6, which causes the computer to choose 5. User chooses 4, which causes the computer to choose 2 in a strong build.

Alternatively,
User chooses 6, which causes the computer to choose 5. User chooses 8, which causes the computer to choose 9 in a strong build.

Alternatively,
User chooses 8, which causes the computer to choose 5. User chooses 2, which causes the computer to choose 4 in a strong build.

Alternatively,
User chooses 8, which causes the computer to choose 5. User chooses 4, which causes the computer to choose 7 in a strong build.

Describe how the rule

```
good(A) :- strong_build(A).
```

succeeds. (2 pts.)

In all of the cases above, the computer cannot win (it only has 1 'x' placed). It also does not need to block a win, due to how I selected my moves. I also selected the moves such that there was not a split which I could get. However, I selected moves such that there was some cell that the computer could choose that would bring it closer to a win (i.e. it is in line with the 'x' already placed), yet, when I block that win, I'm not bringing myself closer to a win (i.e. it is not risky).

Extra Credit

17. If we curry f(x, y, z) into f(x)(y)(z); what does f(x) return? (1 pt.)
    a. Nothing
    b. An integer
    c. A function that takes a single parameter
    d. A pointer

18. Currying allows for more than the required number of parameters (1 pt.)
    a. True
    b. False

19. What does exception handling NOT do? (1 pt.)
    a. Propagate errors up the call stack
    b. Group error types and error differentiation
    c. Separate error handling code from "regular" code
    d. Contain resource leaks from unhandled exceptions

20. Which of the following is not a type of loop improvement? (1 pt.)
    a. Interchange
    b. Prototyping
    c. Induction Variables
    d. Unswitching

21. In call by sharing, what determines whether or not a parameters is passed a value or a reference? (1 pt.)
    Immutable values are call-by-value, while mutable values are call-by-reference.