CS 4501 — LDI — Midterm 1

- **Write your name and UVa ID on the exam.** Pledge the exam before turning it in.
- There are nine (9) pages in this exam (including this one) and six (6) questions, each with multiple parts. Some questions span multiple pages. All questions have some easy parts and some hard parts. If you get stuck on a question move on and come back to it later.
- You have 1 hour and 20 minutes to work on the exam.
- The exam is closed book, but you may refer to your two page-sides of notes.
- Even vaguely looking at a cellphone or similar device (e.g., tablet computer) during this exam **is cheating**.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. We might deduct points if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.
  - *Good Writing Example:* Python and Ruby have implemented some Smalltalk-inspired ideas with a more C-like syntax.
  - *Bad Writing Example:* Im in ur class, @cing ur t3stz!1!
- If you leave a non-extra-credit portion of the exam blank, **you will receive one-third of the points for that small portion (rounded down) for not wasting our time.** If you randomly guess and throw likely words at us, we will be much less sanguine.

UVa ID: KEY

NAME (print): ANSWER KEY

UVa ID: (yes, again!) _____

Problem	Max points	Points
1 — Compiler Stages	11	
2 — OCaml Functional Programming	11	
3 — Python Functional Programming	10	
4 — Regular Expressions	28	
5 — Ambiguity	12	
6 — Earley Parsing	28	
Extra Credit	0	
TOTAL	100	

Honor Pledge:

How do you think you did? _____

1 Compiler Stages (11 points)

Fill in each blank with the number corresponding to the best, most precise answer from the answer bank. Each answer will be used at most once. Each blank is worth one point.

_5_____ Takes as input Regular Expressions.

_12_____ Is produced by Compilers but not Interpreters.

_11_____ Rejects programs that add integers to strings.

_7_____ Rejects programs with unbalanced parentheses.

_9_____ Is used as input by a Lexer.

_2_____ Represents the program structure and expression types.

_3_____ Is used as input by a Parser Generator.

_10_____ Specifies which strings correspond to Tokens.

_8_____ Accepts precedence and associativity annotations.

_1_____ Is used as input by a Type Checker.

_4_____ Produces Tokens as output.

Answer bank:

1. Abstract Syntax Tree
2. Annotated Abstract Syntax Tree
3. Context-Free Grammar
4. Lexical Analyzer (Lexer)
5. Lexical Analyzer Generator
6. Optimizer
7. Parser
8. Parser Generator
9. Program (Cool) Source Code
10. Regular Expressions
11. Semantic Analyzer (Type Checker)
12. Stand-Alone Executable
13. Tokens
14. Typing Rules

2 OCaml Functional Programming (11 points)

Consider the following OCaml functions. The functions are correct and behave as specified; these are all direct copies of standard library functions. (You may use others too!)

```
(* map f [a1; ...; an] applies function f to a1, ..., an, and builds
   the list [f a1; ...; f an] with the results returned by f. *)
let rec map f lst = match lst with
  | [] -> []
  | hd :: tl -> (f hd) :: (map f tl)
(* filter p l returns all the elements of the list l
   that satisfy the predicate p. *)
let rec filter p l = match l with
  | [] -> []
  | hd :: tl -> if p hd then hd :: (filter p tl) else (filter p tl)
(* fold_left f a [b1; ...; bn]] is f (... (f (f a b1) b2) ...) bn. *)
let rec fold_left f accu lst = match lst with
  | [] -> accu
  | hd :: tl -> fold_left f (f accu hd) tl
(* mem a lst is true if and only if a is equal to an element of lst. *)
let rec mem a lst = match lst with
  | [] -> false
  | hd :: tl -> (hd = a) || (mem a tl)
let is_odd n = (n mod 2) = 1 (* returns true if n is odd *)
```

Complete each of the following functions by filling in each blank with a *single* identifier, keyword, operator or constant. You must write well-typed functional programs.

```
(* count_odds returns the # of odd elements in its input list. *)
let count_odds x = _LENGTH_ (_FILTER_ is_odd x)

(* intersect takes two lists as arguments and returns a list of all
   elements they both contain. *)
let intersect a b = filter (fun x -> _MEM_
        _X_ _B_) a

(* reaches takes a list string-string pairs representing the
   directed edges in a graph and returns the set of all nodes
   reachable from node "A" (including "A" itself). *)
let reaches edges =
  let rec helper reached edges = match edges with
  | [] -> reached
  | (a,b) :: tl -> if mem _A_ reached && not (mem _B_ reached)
                   then (helper (_B_ :: reached) edges)
                   else helper _REACHED_ _TL_
  in helper [_"A"_] edges
```

3 Python Functional Programming (10 points)

Consider a function *is_dfa* for determining if a finite state machine description corresponds to a DFA. Recall that a DFA has no epsilon transitions. Recall that a DFA never has two edges leaving the same state with the same label going to different destination states. For example, consider the DFA accepting the regular language denoted by the regular expression $c \mid a(aa)*b$ below:

```
edges = [ ("q0", "a", "q1") ,  # in state q0, on a, goto q1
          ("q0", "c", "q2") ,  # in state q0, on c, goto q2
          ("q1", "b", "q2") ,  # in state q1, on b, goto q2
          ("q1", "a", "q0") ]  # in state q1, on a, goto q0
final = [ "q2" ]
start = "q0"
```

That DFA accepts "ab", "c" and "aaab" but neither "a" nor "aab". However, if we were to add either or both of the following edges:

```
          ("q0", "", "q1") ,  # in state q0 epsilon transition to q2
          ("q1", "b", "q0"),  # in state q1, on b, goto q0
```

the result would not be a DFA. As a reminder, here is the list comprehension syntax:

```
print [x*x for x in range(10) if x > 5] # [36, 49, 64, 81]
```

Complete the following function by filling in each <u>blank</u> with a *single* identifier, keyword, operator or constant. You must write a correct Python program.

```
def is_dfa(edges, final, start):
  any_epsilon = _LEN_([ b for (a,b,c) in edges \
        if _B_ == _""_ ]) > 0

  # Helper function to determine if another edge
  # is leaving the same state on the same character
  # but reaching a different destination.
  def another(a,b,c):
    return len([ x for (x,y,z) in _EDGES_ \\
                if x == _A_ and y == _B_ and \\
                   z _!=_ _C_]) > 0

  edge_problem = len([ a for (a,b,c) in _EDGES_ \\
        if _ANOTHER_(a,b,c) ]) > 0

  return not (any_epsilon or edge_problem)
```

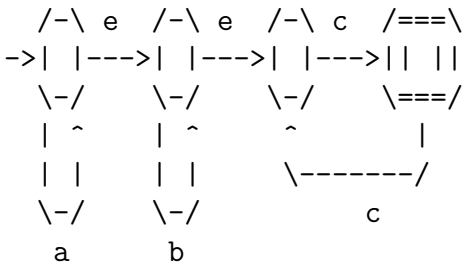
4 Regular Expressions and Automata (28 points)

For this question, the regular expressions are single character (a), epsilon (ϵ), concatenation ($r_1 r_2$), disjunction ($r_1|r_2$), Kleene star $r*$, plus $r+$ and option $r?$.

(a) (7 pts.) Write a regular expression (over the alphabet $\Sigma = \{a, b, c\}$) for the language of strings that have all of the letters in order and an odd number of occurrences of c. Use at most 20 symbols in your answer.

a* b* c(cc)*

(b) (7 pts.) Draw an **NFA** (or DFA) that accepts the language from the above problem. Use at most four states in your answer.



(c) (2 pt.) NEVER. The set of all strings over $\Sigma = \{a, b\}$ that contain more occurrences of a than b is regular.

(d) (2 pt.) ALWAYS. A finite language is both regular and context free.

(e) (2 pt.) SOMETIMES. An infinite language is regular.

(f) (2 pt.) SOMETIMES. Given a DFA d, there exists a regular expression r containing neither $+$ nor $*$ such that $L(d) = L(r)$.

(g) (2 pt.) NEVER. The set of valid Cool programs is context free.

(h) (2 pt.) SOMETIMES. Given an NFA n, there is a DFA d such that $L(d) = \{ss \mid s \in L(n)\}$.

(i) (2 pt.) ALWAYS. Given a DFA d, there is an NFA n such that $L(n) = \{rs \mid r \in L(d) \wedge s \in L(d)\}$.

5 Ambiguity (12 points)

Consider the following grammar G_1.

$$\begin{array}{lcl} S & \rightarrow & E \\ E & \rightarrow & \texttt{int} \\ E & \rightarrow & E + E \mid E - E \\ E & \rightarrow & (E) \end{array}$$

(a) (3 pts.) Show that this grammar is ambiguous using the string "`int - int - int`".

`(int - int) - int`

`vs.`

`int - (int - int)`

(b) (9 pts.) Rewrite the grammar to eliminate left recursion. That is, provide a grammar G_2 such that $L(G_1) = L(G_2)$ but G_2 admits no derivation $X \longrightarrow^* X\alpha$.

$$\begin{array}{lcl} S & \rightarrow & E \\ E & \rightarrow & \texttt{int}\ F \\ E & \rightarrow & (E)\ F \\ F & \rightarrow & +\ E \mid -\ E \mid \epsilon \end{array}$$

6 Earley Parsing (28 points)

(a) (23 pts.) Complete the Earley parsing chart (parsing table) on the next page.

(b) (2 pts.) Give one disadvantage of an LALR(1) parser generator. Give one disadvantage of an Earley parser. Do not exceed four sentences.

LALR(1) only accept a subset of all context-free grammars, may require you to deal with shift/reduce and reduce/reduce conflicts, may require you to rewrite the grammar in an unnatural manner.

Earley can require up to cubic time to parse.

(c) (1 point if not blank.) What's your favorite thing about this class? (If you're in Compilers, answer for Compilers as well.)

(d) (1 point if not blank.) What's your least favorite thing about this class? (If you're in Compilers, answer for Compilers as well.)

(e) (1 point if not blank.) Which "trivia" topics would you like to see discussed during breaks in class?

(f) **Extra Credit (at most 2 points).** Cultural literacy. Below are the English titles of ten important works of world literature or oral tradition. Each work is associated with one of the ten most common languages (by current number of first-language speakers; Ethnologue 2015 estimate). For each work, give the associated language. Be specific.

ENGLISH _____ Jayne Eyre.
HINDI (SANSKRIT) _____ Lake of the Deeds of Rama.
SPANISH _____ One Hundred Years of Solitude.
ARABIC _____ One Thousands and One Nights.
RUSSIAN _____ The Brothers Karamazov.
PORTUGUESE _____ The Crime of Father Amaro.
MANDARIN (CHINESE) _____ The Dream of the Red Chamber.
JAPANESE _____ The Tale of Genji.
JAVANESE (MALAYSIAN)_____ Wayang Kulit.
BENGALI _____ Where the Mind is Without Fear.

Grammar

T → E + E + E
E → int | (A
A → int A |) | ε

Input

int + (int) + int

int	+	(	int	)	+	int

chart [0]	chart [1]	chart [2]	chart [3]	chart [4]	chart [5]	chart [6]	chart [7]
T→•E+E+E,0	E→int•,0	T→E+•E+E,0	E→(• A,2	A→int• A,3	A→)•,4	T→E+E+•E,0	E→int•,6
E→•int,0	T→E•+E+E,0	E→•int,2	A→•int A,3	A→•int A,4	A→int A•,3	E→•int,6	T→E+E+E•,0
E→•(A,0		E→•(A,2	A→•),3	A→•),4	E→(A•,2	E→•(A,6	
			A→•,3	A→•,4	T→E+E•+E,0		
			E→(A•,2	A→int A•,3			
			T→E+E•+E,0	E→(A•,2			
				T→E+E•+E,0			