

First Midterm Exam
CS164, Spring 2007
February 26, 2007

- Please read all instructions (including these) carefully.
- Write your name, login, and SID.
- No electronic devices are allowed, including cell phones used as watches.
- Silence your cell phones and place them in your bag.
- The exam is closed book, but you may refer to one (1) page of handwritten notes.
- Solutions will be graded on correctness and **clarity**. Each problem has a relatively simple and straightforward solution. Partial solutions will be graded for partial credit.
- There are 7 pages in this exam and 6 questions, each with multiple 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.
- 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. **Do not** use any additional scratch paper.

LOGIN: _____

NAME: _____

SID: _____

Problem	Max points	Points
1	17	
2	24	
3	20	
4	15	
5	24	
6	15	
TOTAL	115	

Problem 1: Prototype-Based Programming [17 points]

Part 1. [5 points] John first writes this JavaScript program.

```
var o = {f:1}
function foo(x,y) { console.log(x.f+y) }
foo(o,2)
```

After he is satisfied that the program correctly outputs “3”, he decides to make the program “object-oriented.” Specifically, he wants to turn **o** into a receiver of function **foo**. He rewrites the program as follows.

add your new code below this line

```
var o = {f:1}
function foo(y) { console.log(this.f+y) }
o.foo(2)
```

Unfortunately, the new program does not behave as the original program. Correct it by **adding** a single assignment. The fixed program must behave as the original one.

Part 2. [12 points] The following code creates four objects, shown in the figure.

- (1) Draw edges showing values of the `__proto__` properties in these four objects.
- (2) Mark with “C” `__proto__` properties that are read during the call `c.parentMethod()`.
- (3) Mark with “P” `__proto__` properties read during the call `p.parentMethod()`.

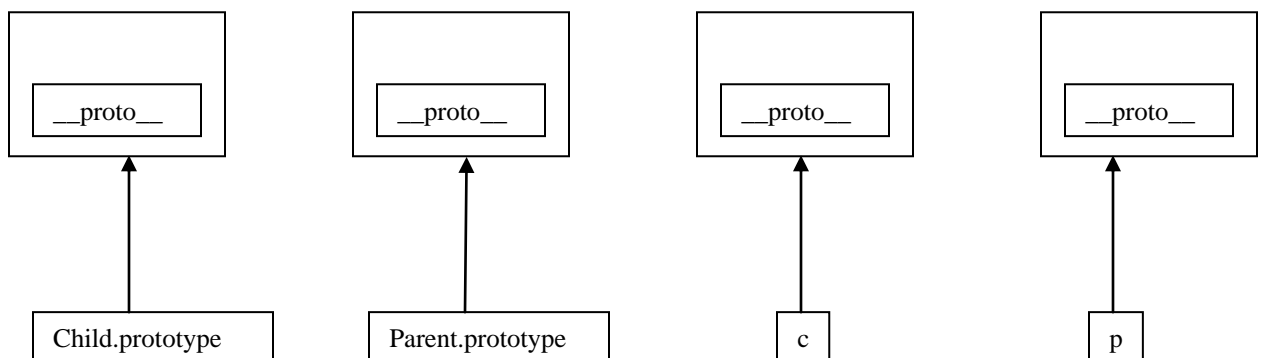
Note that the two calls read both the property **parentMethod** and the property **instVar**.

```
function Parent() { this.instVar = 1; }
Parent.prototype = { parentMethod: function() { return this.instVar; } }
function Child() { }
```

```
Child.prototype = new Parent;
Child.prototype.parentMethod = function() { return this.instVar+1; }
```

```
var p = new Parent
var c = new Child
```

```
p.parentMethod()
c.parentMethod()
```



Problem 2: Finite Automata and Regular Expressions [24 points]

In this question, you are asked to write a finite automaton AND a regular expression for two regular languages. You can choose either a deterministic or a non-deterministic automaton. *Note that you will be penalized for using an excessive number of states.* You can use regular expression operators from the lecture, or from JavaScript or from Python.

Language 1: *Strings with an even number of zeros and an even number of ones.*

Automaton [6 points]: This problem is similar to one from the discussion section notes.

Regular Expression [6 points] **Hint:** Think of each pair of characters 00, 01, 10, and 11 as a single input character.

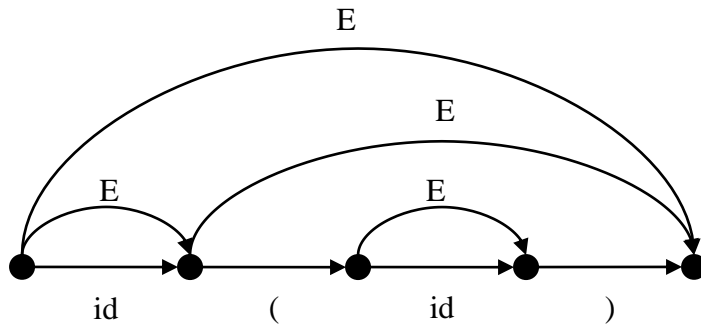
Language 2: *JavaScript's C-style comments. A comment starts with /* and ends with */. Comments cannot be nested.* Note: Assume that the alphabet is {/, *, a-z, "}. Two strings from the language: /* "hello" is a string */; /* this is /* one comment */. A string not from the language: /* /* nesting not allowed */ */. Careful, the comment ends after the first */.

Automaton [6 points]:

Regular Expression [6 points]:

Problem 3: CYK Parser [20 points]

Part 1. [8 points] Assume that the CYK parser executed on the input “id (id)” produced the following (and only the following) reductions. Note that the parser is not restricted to productions with one or two symbols on the right-hand side.



Consider the four grammars below. Which grammar(s) may have produced the parse result shown above? Mark the corresponding letter(s) by filling their circles:

☐ A ☐ B ☐ C ☐ D

$E \rightarrow id \mid id (E)$	$E \rightarrow id \mid id (E) \mid (E)$	$E \rightarrow id \mid E (id) \mid (E)$	$E \rightarrow id \mid E (E)$
grammar A	grammar B	grammar C	grammar D

Part 2. [12 points] We propose to modify the CYK algorithm by having it terminate as soon as the edge (start, end, S) appears in the parse graph.

- Will this modified CYK parser recognize the same language as the original one?
- Give a grammar and an input for which this modified CYK may---depending on the order in which edges are processed---take substantially less time than the original parser.
- Why might this modification be a bad idea?

Problem 4: Recursive Descent Parser [15 points]

Part 1. [9 points] This question asks you to identify techniques useful or essential with recursive descent parsers.

1. When your recursive descent parser performs too much backtracking, which technique(s) listed below will you apply? **Answer:** OA OB OC OD
Justification:
2. When the grammar causes non-termination of your recursive descent parser, which of technique(s) listed will you apply? **Answer:** OA OB OC OD
Justification:
3. When your recursive descent parser always terminates but fails to accept a string that *is* in the language of the grammar that the parser intends to encode, which technique(s) may help? **Answer:** OA OB OC OD **Justification:**

- A Removing Ambiguity
- B Left-Recursion Elimination
- C Left Factoring
- D Reordering Productions

Part 2. [6 points] Consider the following grammar

$$E \rightarrow \text{id} \mid \text{id} + E$$

Also consider this recursive descent parser.

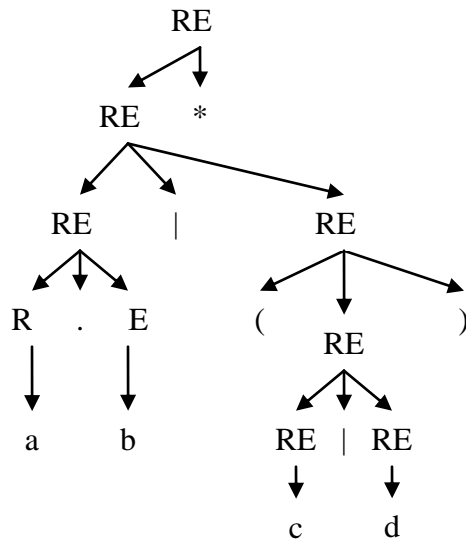
```
bool main() { return E() && term EOF; }
bool E() {
    int save = next; // save input position
    return E1() || (next=save, E2());
}
bool E1() { return term ID; }
bool E2() { return term ID && term '+' && E(); }
```

Give a string that that is in the grammar but is not accepted by the parser: _____

Problem 5: Grammars [24 points]

Part 1: [8 points] Draw an AST that represents the regular expression $a.b|(c|d)^*$. The AST must correctly represent the precedence of operators in the regular expression. Note that the regular expression operator '.' stands for concatenation.

Part 2: [8 points] Write a grammar that accepts only legal regular expressions. You need to support operators '.', '|', '*', and parentheses. The grammar should be ambiguous in that it must ignore the precedence of the operators; for example, on the input $a.b|(c|d)^*$, the grammar should be able to produce also this parse tree:



Part 3: [8 points] Disambiguate your grammar from Part 2 in such a way that its parse tree can be used to construct the AST from Part 1. You must rewrite the grammar; do not use disambiguation declarations.

Part 1 (AST)	Part 2: Ambiguous Grammar	Part 3: Disambiguated Grammar

Problem 6: Earley Parser [15 points]

Consider the grammar

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow T + E \mid T \\ T &\rightarrow T * \text{int} \mid \text{int} \end{aligned}$$

Assume the Earley parser is given the input `int+int*int`. Draw the first ten (10) edges placed by the parser on this input. Assume that the parser inserts in-progress edges only when no other edges can be placed (this restriction will make the answer unique).

Draw not only the completed edges but also the predicted and the in-progress edges. Draw the edge labels legibly. Indicate the order in which the edges were added. For example, the fifth edge may be labeled as follows:

5: $E \rightarrow T . + E$

