Midterm I

- Please read all instructions (including these) carefully.
- There are six questions on the exam, each worth between 15 and 30 points. You have 3 hours to work on the exam.
- The exam is closed book, but you may refer to your four sheets of prepared notes.
- 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. There are no "tricky" problems on the exam—each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary.

NAME: Sample solution

SID or SS#: _____

Problem	Max points	Points
1	15	
2	15	
3	20	
4	20	
5	30	
6	30	
TOTAL	130	

1. **Regular Expressions** (15 points) The following description of integer constants in C++ appears on page 480 of "The C++ Programming Language."

> An integer constant consisting of a sequence of digits is taken to be decimal (base ten) unless it begins with 0 (digit zero). A sequence of digits starting with 0 is taken to be an octal integer (base eight). The digits 8 and 9 are not octal digits. A sequence of digits preceded by 0x or 0X is taken to be a hexadecimal integer (base sixteen). The hexadecimal digits include a or A through f or F .... The type of an integer depends on its ...suffix. [An integer may have no suffix.] ...If [the integer] is suffixed by u or U, its type is .... If it is suffixed by l or L, its type is .... [Either or both suffixes may appear at most once each in either order.]

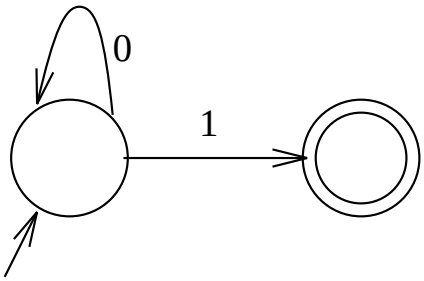
Give a regular expression for C++ integer constants. Use *only* the operations $+$, $*$, and concatenation in your expression (no special flex notation). You may also use parentheses and ϵ (which are not operations). For convenience, you may name a regular expression R by writing $name = R$.

```
[spaces are not significant]

octal = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7
nonzero = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
decimal = 0 + nonzero
hex = decimal + a + A + b + B + c + C + d + D + e + E + f + F

optu = u + U + epsilon
optl = l + L + epsilon
suffix = (optu optl)+(optl optu)

number = (octal octal*) + (nozero decimal*) + ((0x + 0X) hex hex*)
integer = number suffix

[Note: We also allowed (0x hex*)]
```

2. **Finite Automata and Regular Expressions** (15 points)

For each of the following finite automata, give an equivalent regular expression. Clearly mark your answer to each part. As in the previous problem, use only the basic regular expression notation.

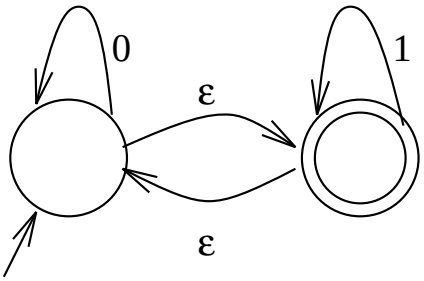
(a)



5 points

RE: 0*1

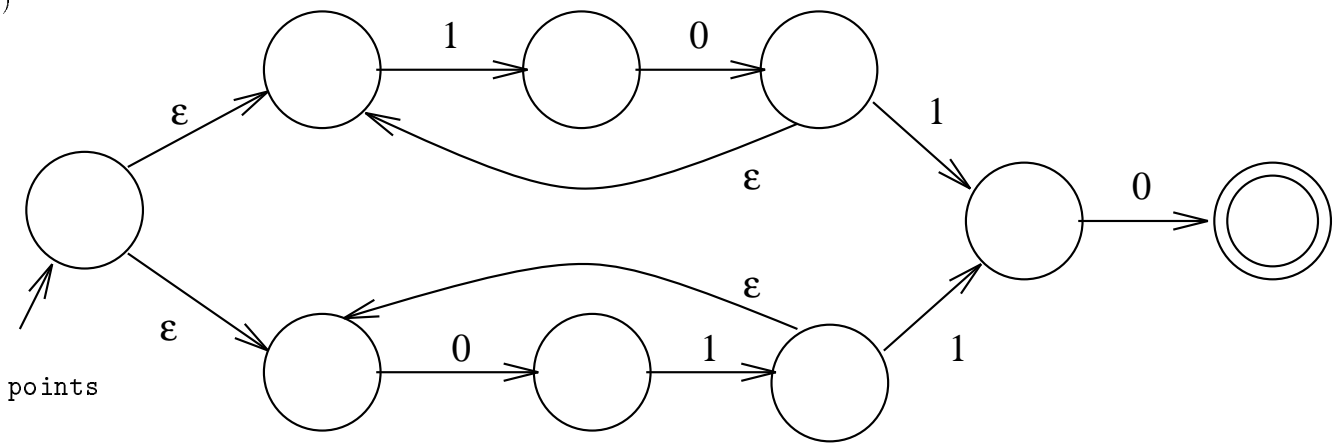
(b)



5 points

RE: (0 + 1)*

(c)



5 points

RE: ((10 (10)*) + (01 (01)*)) 10

3. **Grammars** (20 points)

Give a context-free grammar for each of the following languages.

(a) Any string of x's followed by an equal number of y's.

7 points

```
S -> xSy | epsilon
```

(b) Any string of x's of length n followed by a string of y's of length $2n + 1$, for any $n \geq 0$.

7 points

```
S -> xSyy | y
```

(c) Any string of x's and y's with an equal number of x's and y's. The x's and y's may appear in any order.

6 points

```
S -> xSyS | ySxS | epsilon
```

4. **Top-Down Parsing** (20 points)

Consider the following grammar for a subset of English sentences. The nonterminals are S, NP, VP, AP. The terminals are `the, noun, adjective, verb`. The start symbol is S.

$$
\begin{array}{rcl}
\texttt{S} & \rightarrow & \texttt{NP VP} \\
 & | & \texttt{NP VP NP} \\
\texttt{NP} & \rightarrow & \texttt{the AP noun} \\
\texttt{AP} & \rightarrow & \texttt{AP adjective} \\
 & | & \epsilon \\
\texttt{VP} & \rightarrow & \texttt{verb}
\end{array}
$$

Give an LL(1) grammar that generates the same language as this grammar. Show the parsing table for the grammar you produce.

a) Left factor

$$
\begin{array}{rcl}
\texttt{S} & \rightarrow & \texttt{NP VP NP}_{\texttt{opt}} \\
\texttt{NP}_{\texttt{opt}} & \rightarrow & \texttt{NP} \\
 & | & \epsilon \\
\texttt{NP} & \rightarrow & \texttt{the AP noun} \\
\texttt{AP} & \rightarrow & \texttt{AP adjective} \\
 & | & \epsilon \\
\texttt{VP} & \rightarrow & \texttt{verb}
\end{array}
$$

b) Eliminate left recursion

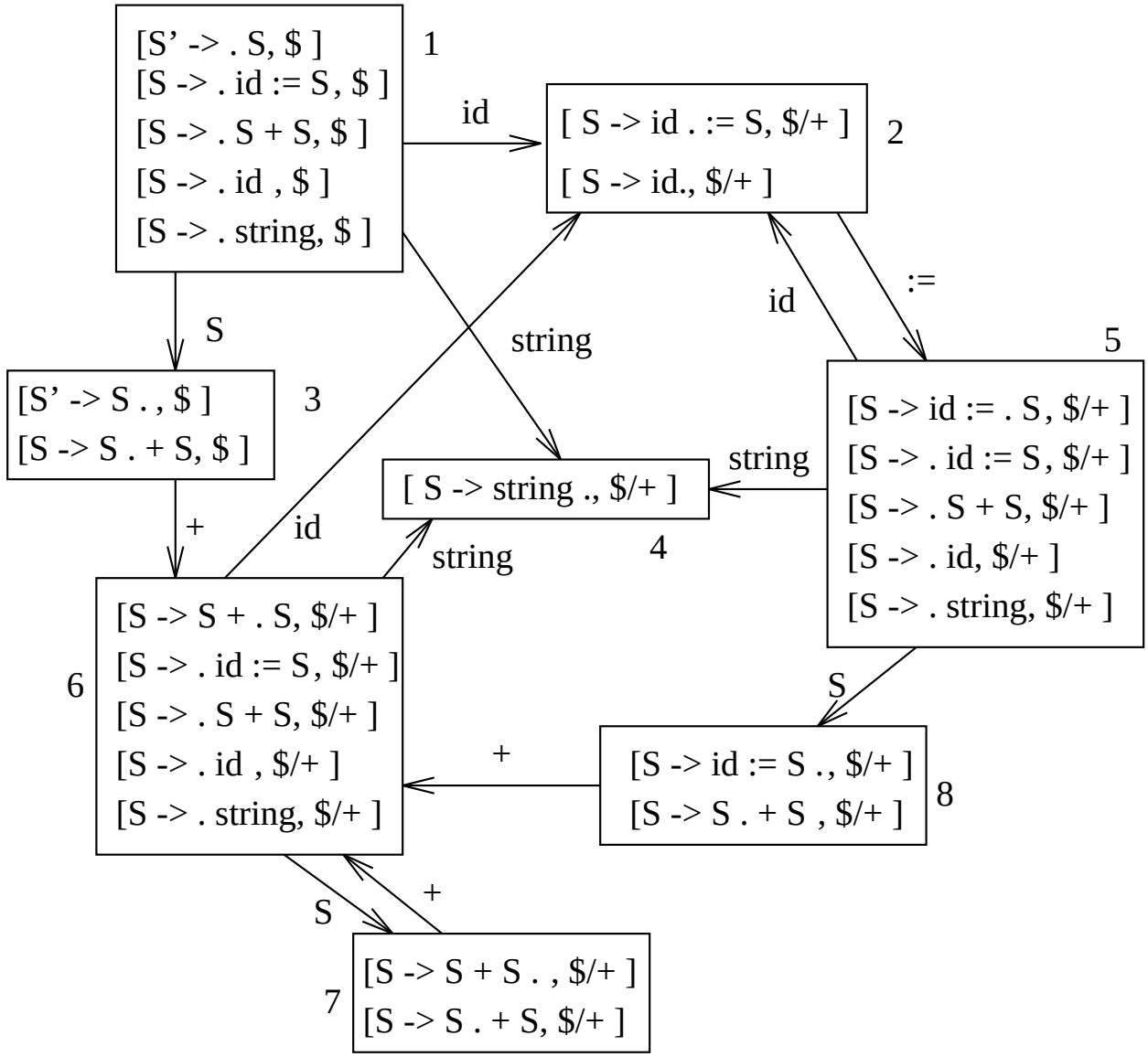
$$
\begin{array}{rcl}
\texttt{S} & \rightarrow & \texttt{NP VP NP}_{\texttt{opt}} \\
\texttt{NP}_{\texttt{opt}} & \rightarrow & \texttt{NP} \\
 & | & \epsilon \\
\texttt{NP} & \rightarrow & \texttt{the AP noun} \\
\texttt{AP} & \rightarrow & \texttt{adjective AP} \\
 & | & \epsilon \\
\texttt{VP} & \rightarrow & \texttt{verb}
\end{array}
$$

c) Construct table

	the	noun	verb	adjective	$
S	$\texttt{S} \rightarrow \texttt{NP VP NP}_{opt}$				
$\texttt{NP}_{opt}$	$\texttt{NP}_{opt} \rightarrow \texttt{NP}$				$\texttt{NP}_{opt} \rightarrow \epsilon$
NP	$\texttt{NP} \rightarrow \texttt{the AP noun}$				
AP		$\texttt{AP} \rightarrow \epsilon$		$\texttt{AP} \rightarrow \texttt{adjective AP}$	
VP			$\texttt{VP} \rightarrow \texttt{verb}$		

5. **LALR Parsing and Conflicts** (30 points)

Consider the following DFA of LR(1) items for a LALR(1) parser. The states are numbered; refer to these numbers where appropriate in the following questions.



(a) Give the grammar used in constructing the DFA.

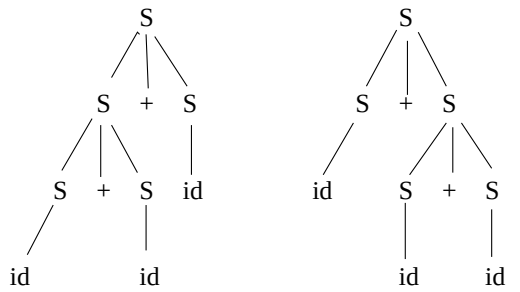
5 points

```
S -> id | string | S + S | id := S
```

(b) Show that the grammar is ambiguous.

5 points

The string id + id + id has two parse trees:



(c) Which of the following strings are viable prefixes? For each string that is a viable prefix, give a suffix that produces a right sentential form.

i. S

2 points

Yes: suffix: $

ii. + S +

1 point

No.

iii. S + id :=

2 points

Yes: suffix: id $

(d) Which states of the DFA have conflicts and of what kind (shift-reduce or reduce-reduce)? Be sure to state clearly the cause of the conflict in each case.

5 points

State 7 has a shift-reduce conflict on input +.

State 8 has a shift-reduce conflict on input +.

(e) Which conflicts should be resolved in which way to give the `:=` operator higher precedence than `+`?

`5 points`

`Reduce should be chosen in state 8.`

(f) Which conflicts should be resolved in which way to cause the `+` operator to associate to the right?

`5 points`

`Shift should be chosen in state 7.`

6. **Error Recovery** (30 points)
This problem uses the same DFA as the previous problem. You will need to refer to that DFA again to answer the parts of this question. Assume that any shift/reduce conflicts are resolved in favor of reducing.

(a) Consider the string

`id := string string`

Show the sequence of configurations of the LALR(1) parsing algorithm as it parses this string according to the DFA of the previous problem. Indicate exactly where the error is detected.

```
10 points

(# id := string string)
(id # := string string)
(id := # string string)
(id := string # string)

Error: cannot reduce.
```

(b) Consider a new action `panic N t`, where `N` is a non-terminal and `t` is a terminal. The meaning of the action is to push `N` on the stack and discard all input tokens up to, but not including, the next `t`. If there is no next `t`, then `N` is pushed on the stack and all remaining input is discarded. Assume we want the parser to take action `panic S +` whenever it is in configuration `S` + #`w` and the first token of `w` is erroneous. Which entries in the `action` table should have `panic S +`? (You do not need to give the entire `action` table to answer the question; just indicate which entries have the new action.) Show the sequence of configurations of the parser on input

`id+ := id + +id`

```
10 points

action(6, :=) = panic S +
action(6, +) = panic S +
action(6, $) = panic S +

(#id + := id + + id)
(id # + := id + + id)
(S # + := id + + id)
(S  + # := id + + id)
(S + S # + + id)
(S # + + id)
(S + # + id)
(S + S # + id)
(S # + id)
(S + # id)
(S + id #)
```

```
(S + S #)
(S #)
(S' #)
```

(c) Consider a new action `insert t` where `t` is a terminal. The meaning of the action is to insert `t` before the next input token (i.e., `t` becomes the next input). Give `insert` actions that implement each of the following local recovery strategies. Be sure to say which entries in the `action` table have the `insert` actions.

i. If the input contains an `id` followed by a `string`, insert an `:=` between them.

```
action(2, string) = insert :=
```

ii. If the input contains two consecutive strings, insert a `+` between them.

```
action(4, string) = insert +
```

iii. If the input contains a `string` followed by an `id`, insert a `+` between them.

```
action(4, id) = insert +
```