MIT 6.035 Spring 2011 Quiz 1 (100 points)

Your Full Name Here:

Your Athena ID Here:

1. (5 points) Write a regular expression for the language $L = \{0^n 1^m \mid (n + m) \text{ is even}\}$.

$(00)^*(11)^* \mid 0(00)^*1(11)^*$

2. (20 points) Let the alphabet $\sum = \{0, 1\}$.

(a) (5 points) Write a regular expression for the language of all strings over $\sum$ that contain the contiguous substring 11.

$(0 \mid 1)^* 11 (0 \mid 1)^*$

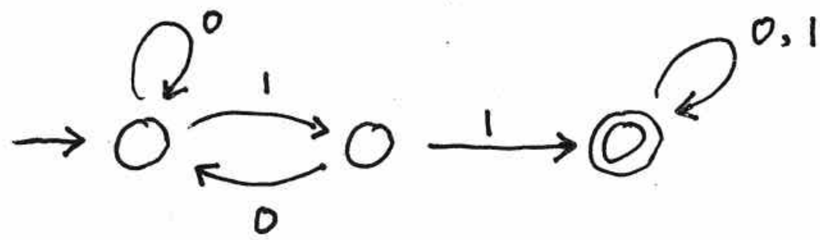
(b) (5 points) Write a regular expression for the language of all strings over $\sum$ that do not contain the contiguous substring 11.

$0^* \mid 0^* 1 (00^* 1)^* 0^*$

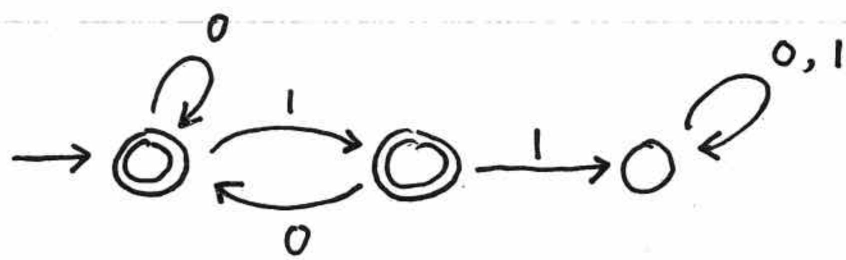
or

$(10 \mid 0)^* (1 \mid \varepsilon)$

(c) (5 points) Give a non-deterministic finite automaton (NFA) for the language of all strings over $\sum$ that contain the contiguous substring 11.

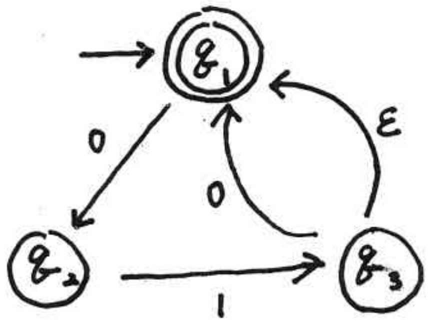


(d) (5 points) Give a non-deterministic finite automaton (NFA) for the language of all strings over $\sum$ that don't contain the contiguous substring 11.

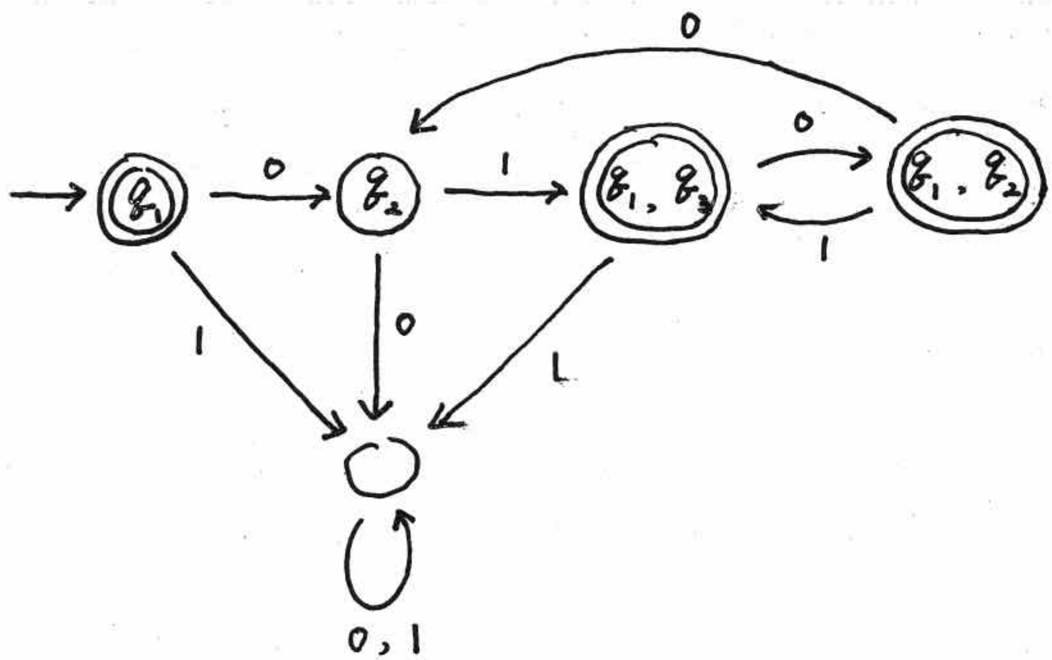


3. (10 points)

(a) (5 points) Give a non-deterministic finite automaton (NFA) for the language L = (010 | 01)*. The NFA must contain at most 3 states. (Hint: draw an NFA with 4 states, then optimize).



(b) (5 points) Give a deterministic finite automaton for the language L.



4. (30 points)

Consider the following grammar:

$S \to L = R$

$L \to *R \mid id$

$R \to L$

You can think of L and R as standing for l-value and r-value, respectively. * is the dereference operator or indirection operator in C-like languages.

A shift-reduce parser can perform the following sequence of actions to accept the string "*id = id".

shift » shift » reduce » reduce » reduce » shift » shift » reduce » reduce » reduce » accept

(a) (10 points) Give a sequence of actions that a shift-reduce parser can take to accept the string "id = id".

shift → reduce → shift → shift → reduce → reduce → reduce → accept

(b) (10 points) Give a sequence of actions that a shift-reduce parser can take to accept the string "*id = *id".

shift → shift → reduce → reduce → reduce → shift → shift → shift → reduce → reduce → reduce → reduce → reduce → accept

(c) (10 points) Is the grammar ambiguous? Why or why not?

No.

There doesn't exist a string which can be generated by the grammar in more than one way.

5. (15 points)

Consider the following grammar:

S → `if` E `then` S `else` S | `begin` S L | `print` E | ε

L → `end` | `;` S L

E → num = num

The goal is to write a recursive-descent parser for the grammar. You are given the following `L()` and `E()` functions. Your job is to write the `S()` function on the next page.

```
L() {
      if (token = end) {
            match(end);
      } else if (token = ;) {
            match(;); S(); L();
      } else {
            throw SyntaxError;
      }
}

E() {
      if (token = num) {
            match(num); match(=); match(num);
      } else {
            throw SyntaxError;
      }
}
```

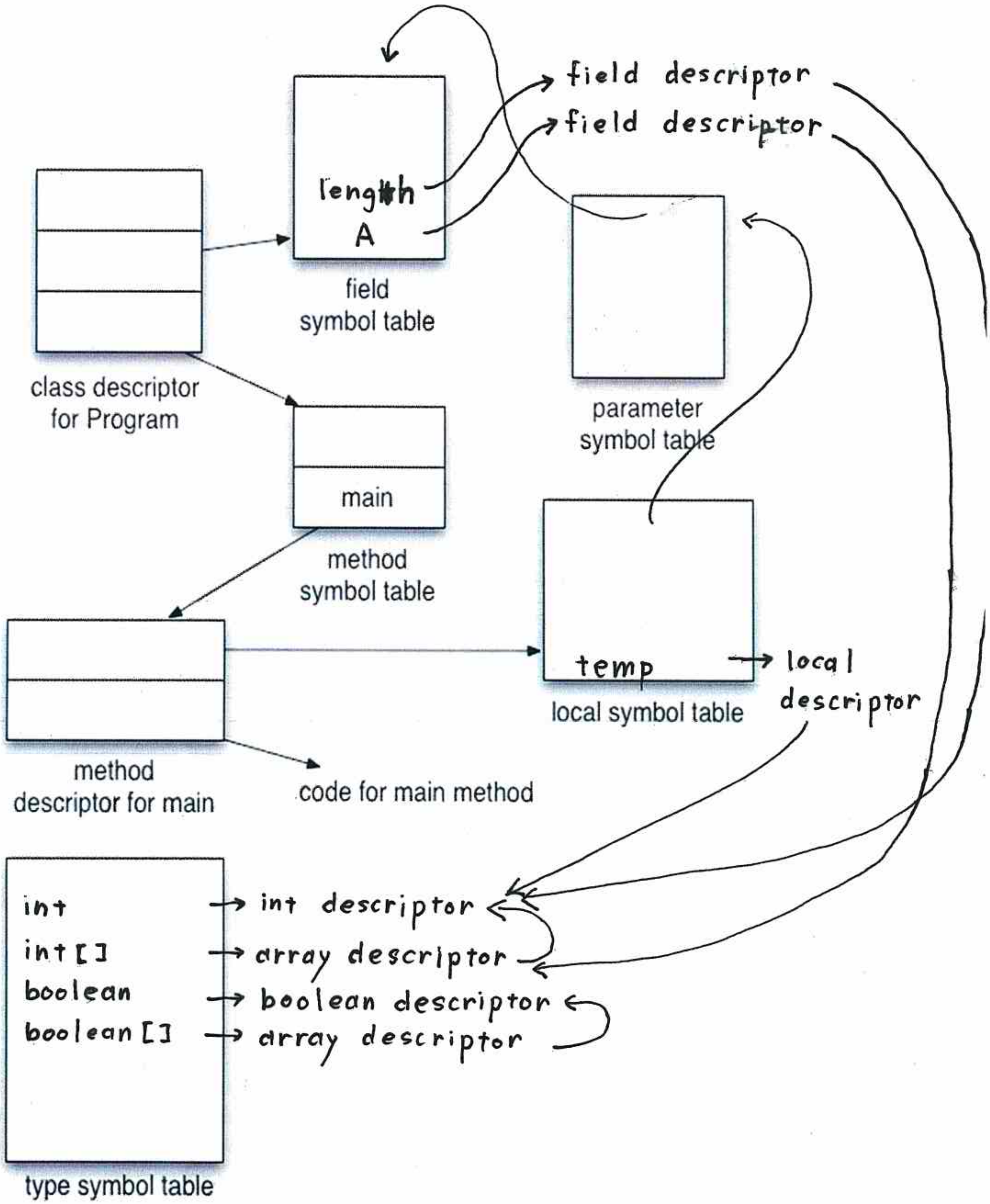
```
S() {
    if (token = if) {
        match(if); E(); match(then); S();
        match(else); S();
    } else if (token = begin) {
        match(begin); S(); L();
    } else if (token = print) {
        match(print); E();
    } else { // S → ε
        // do nothing
    }
}
```

6. (20 points)

The following is a code snippet of `legal-01.dcf`:

```
class Program {
      int A[100];
      int length;

      void main() {
            int temp;

            length = 100;

            callout("srandom", 17);

            for i = 0, length {
                  temp = callout("random");
                  A[i] = temp;
            }

            /* <HERE> */

      }
}
```

What should the symbol tables look like at <HERE>, considering the semantics of the Decaf language? Complete the symbol tables on the next page in the similar way to the symbol tables presented at Lecture 5. (Hint: note that the Decaf language is different from the language presented at Lecture 5).


field descriptor
field descriptor
length
A
field
symbol table
parameter
symbol table
class descriptor
for Program
main
method
symbol table
temp
local
descriptor
local symbol table
method
descriptor for main
code for main method
int
int[]
boolean
boolean[]
int descriptor
array descriptor
boolean descriptor
array descriptor
type symbol table