Name: ____

You should submit 1. a hard-copy of pages 1–5 of this assignment with your answers filled in, and 2. an electronic package that contains the source files for your work on the programming questions, by following the submission procedure described in class and on the class newsgroup. You are welcome to use any inanimate resources (e.g., books, Web sites, publicly available code) to help you with your work. However, *all such help must be clearly noted* in your submissions. Further, no matter what you use, *you must be able to explain, in detail, how it works.* (You may be called upon to explain your homework in person.) Refer to the class policy for details, and ask for clarifications if you are unsure if something is allowed.

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) Package and submit your solutions to the programming questions via `http://cs.umaine.edu/~chaw/u/`. *After* submitting your work, fill in the following:

File name:	______________________
Size, in bytes:	______________________
MD5 checksum:	______________________
Timestamp:	______________________

3. (3 pt.) Post a *useful* message on the class newsgroup (e.g., well written questions, answers, observations, and pointers to resources.)

4. (45 pts.) Provide DFAs for each of the following languages. Depict the automata graphically in addition to providing their formal definitions.

   (a) Strings over the alphabet $\{1, 2, 3\}$ that contain either 112, 21223, or 312 as substrings.

(b) Strings over the alphabet $\{a, b\}$ in which every occurrence of a is followed immediately by three consecutive bs.

(c) Binary strings in which the number of 1s is a multiple of 7.

5. (50 pts.) Provide a *well documented and explained* implementation of the *self* program discussed in class (and page 247 of the textbook), keeping the restrictions in mind, in C, Java, Python, or Scheme. Try to follow the template as closely as possible. Be sure to clearly identify the parts of your code that correspond to the modules A and B of that program. Identify the output from A, how it is used by B, and the output from B.

[additional space for answering the earlier question]

6. (100 pts.) Implement an interpreter for the language *Lexaard* (language for exploring automata and related doodads), outlined below. The description covers the main points but is not exhaustive. Use discussions in class and on the class newsgroup for clarifications and further details.

   You should submit a well documented source code package that includes a Makefile and README file with the conventional contents. Your submission should yield an executable named `lexaard` which reads from standard input and writes to standard output.

   The language consists primarily of newline-terminated statements, with exceptions noted below. Each statement, and each line of the input, consists of whitespace-separated tokens, where whitespace is a nonempty sequence of any mix of spaces and tabs. Whitespace at the beginning and end of a line is permitted but not required. The first token of each statement is a *verb* that determines how the rest of the statement is interpreted. The language uses only an easily printable subset of the 7-bit ASCII character set (letters, digits, punctuation, space, tab, newline) and is case sensitive.

   The language has three types of objects: symbols, strings, and automata. Symbols are unquoted strings (sequences of characters) that follow the typical rules for identifiers in a language such as Java or C. Examples: `x`, `m101`, `my_first_automaton`. Strings use the familiar quoted representation. Examples: `"x"`, `"am I a string?"`. Automata are represented as suggested by the following two equivalent representations of the automaton M_1 from page 36 of the textbook. (For clarity, we use ␣ to denote a space character. There is a single newline character terminating each line, and a single blank line that terminates each representation.)

```
fsa
m101
␣␣␣␣␣0␣␣␣␣1
␣q1␣␣q1␣␣␣q2
*q2␣␣q3␣␣␣q2
␣q3␣␣q2␣␣␣q2

```

```
fsa
␣␣␣m101␣a␣rather␣pointless␣comment
0␣1
␣␣␣␣q1␣q1␣q2␣␣␣
␣␣*q2␣␣␣␣␣q3␣␣␣␣␣␣␣␣␣␣q2␣␣
q3␣q2␣␣␣q2

```

   An FSA's representation always begins with the literal `fsa` followed by a newline. The first token on the next line (`m101` above) is a descriptive identifier associated with the automaton. Any further tokens on this line are ignored. The next line lists the alphabet of the automaton ($\{\texttt{0}, \texttt{1}\}$ above). These lines are followed by one line for each state of the automaton (three lines for states `q1`, `q2`, and `q3` above). The state listed first (`q1`) is the start state of the automaton. An accepting state is denoted by adorning its line with a `*` prefixed to the state's name in the leftmost column. The representation suggested above is intentionally very similar to the usual tabular description of an automaton's state-transition table, such as the one on page 36 of the textbook. For example, $\delta(\texttt{q3}, \texttt{1}) = \texttt{q2}$ above. In both input and output, it is good style to format an automaton's representation as on the left above to ease its tabular interpretation. However, such formatting is not required, and the representation on the right yields an identical automaton.

The interpreter must produce output exactly when and as described below for each statement. In particular, it must not produce extraneous output such as prompts and informative feedback unless noted below. The descriptions use `typewriter font` for literal text and *italic font* for meta-variables.

`quit` Exit the interpreter completely. The interpreter must also exit at the end of standard input.

`print` ***x*** Print the external representation of the object named *x*. It is not an error if *x* is undefined; print nothing in this case. Automata should be printed in a well-formatted manner (but it is not strictly incorrect to print them in a different, valid manner).

`define` ***x v*** Define the name *x* to be the object represented by *v*.

`run` ***x i*** Run the automaton named *x* on the input string literal *i*. It is an error if *x* is not defined to be an automaton. The output is a single line containing `accept` or `reject` depending on whether the automaton accepts or rejects the input.

`run` ***x n*** As above, except *n* is the name of a previously defined string that is used as input to the automaton.

Blank lines, i.e., lines composed of only whitespace, are ignored, except when they are used in representations of objects. For this submission, you may assume that all test input will be valid; however, you are encouraged to implement at least rudimentary error checking.

```
define␣x␣"01011"
print␣x
define␣x␣"1101011"
print␣x
define␣m1␣fsa
m1orwhatever
0␣␣␣1
q1␣␣q1␣␣q2
*q2␣␣q1␣␣q2

print␣m1
run␣m1␣"000101010010"
run␣m1␣"0001010100101"
run␣m1␣"0001010100100"
run␣m1␣x
quit
```

```
01011
1101011
m1orwhatever
␣␣␣␣␣␣0␣␣1
␣q1␣␣q1␣␣q2
*q2␣␣q1␣␣q2

reject
accept
reject
accept
```

Figure 1: Sample input (left) and output (right).