Source File: `~/4301/04/lab04.(C|CPP|cpp|c++|cc|cxx|cp)`
Input: Under control of `main` function
Output: Under control of `main` function
Value: 1

For $\Sigma = \{a, b, c\}$, construct a finite state automata that accepts the set consisting of

$$\left\{ \begin{array}{l} x | x \in \{a, b, c\}^+ \text{ and} \\ \quad \text{the number of } a\text{'s plus the number of } b\text{'s plus twice the number of } c\text{'s is divisible by six.} \end{array} \right\}$$

Valid strings include `aaaaaa`, `bbbbbb`, `ccc`, `babac`, `caabb`, `bcbaa`, `abcba`, `ccaa`, `cabc`, `cccccc`, `aacbbccc`, `ccccccccc`, `aaaaaaaaaaaabbbbbbbbbbbb`, and `aaabbb` but not ϵ, `a`, `b`, `aa`, `ab`, `ba`, `bb`, `aaa`, `bbb`, `abba`, `abcb`, or `abbbbaa`.

A header file is shown in Figure 1, a sample `main` function for testing your implementation is shown in Figure 2, and a sample execution sequence is shown in Figure 3. To use the `Makefile` as distributed in class, add a target of `lab04` to `targets2srcfiles`.

Additional notes:

- The `main` function appends the character '`%`' to each line as it is read in.
- The halt state is `0` and the start state is `1`.

```
1  #ifndef FSA_H
2  #define FSA_H
3
4  #include <iostream>
5  #include <string>
6  #include <map>
7
8  using namespace std;
9
10 class TableEntry
11 {
12  public:
13   TableEntry(char ch, uint state)
14   {
15     setInputCharacter(ch);
16     setNextState(state);
17   }
18   void setInputCharacter(char ch)
19   {
20     inputCharacter = ch;
21   }
22   void setNextState(uint state)
23   {
24     nextState = state;
25   }
26   char getInputCharacter() const
27   {
28     return inputCharacter;
29   }
```

Figure 1. `/usr/local/4301/include/fsa.h` (Part 1 of 2)

```
30   private:
31    char inputCharacter;
32    uint nextState;
33  };
34
35  class FSA
36  {
37    uint getNextState() const
38    {
39      return nextState;
40    }
41   public:
42    // default constructor -- initializes private data members name,
43    // labNumber, and description
44    FSA();
45    // Member function initializeMachine() initializes the private data
46    // member machine, a multimap where the key is the current state and
47    // the value is a class object containing the input character and
48    // the next state
49    void initializeMachine();
50    // Member function outputID() writes name, class, lab number, and
51    // lab description to output stream out
52    void outputID(ostream& out) const;
53    // Member function implementFSA() returns true if dataLine is
54    // recognized by the FSA as valid and false otherwise
55    bool implementFSA(string dataLine) const;
56   private:
57    string name;
58    int labNumber;
59    string description;
60    multimap<uint, TableEntry> machine;
61  };
62
63  #endif
```

Figure 1. `/usr/local/4301/include/fsa.h` (Part 2 of 2)

```
1  #include <fsa.h>
2
3  using namespace std;
4
5  int main()
6  {
7    FSA myFSA;
8    string dataLine;
9
10   myFSA.initializeMachine();
11   myFSA.outputID(cout);
12
13   while (getline(cin, dataLine))
14   {
15     cout << "Input:  " << dataLine;
16     dataLine += "%";
17     if (myFSA.implementFSA(dataLine))
18       cout << " *** valid input ***" << endl << endl;
19     else
20       cout << " --- invalid input ---" << endl << endl;
21   }
22
23   return 0;
24 }
25
26 void FSA::outputID(ostream& out) const
27 {
28   out << name << endl;
29   out << "CS 4301" << endl;
30   out << "Lab " << labNumber << endl;
31   out << description << endl << endl;
32 }
33
```

Figure 2. `/usr/local/4301/src/lab04main.C` (Part 1 of 2)

```
34 bool FSA::implementFSA(string dataLine) const
35 {
36   int currentState = 1;
37   string::iterator dataItr = dataLine.begin();
38   multimap<uint, TableEntry>::const_iterator fsaItr;
39
40   while (currentState > 0)
41   {
42     // Use find to return an iterator to the first entry with a key of
43     // currentState
44     fsaItr = machine.find(currentState);
45     if (fsaItr != machine.end()) // found a key of currentState
46     {
47       while (fsaItr != machine.upper_bound(currentState) &&
48              fsaItr->second.getInputCharacter() != *dataItr)
49         ++fsaItr;
50       if (fsaItr != machine.upper_bound(currentState))
51       {
52         currentState = fsaItr->second.getNextState();
53         ++dataItr;
54       }
55       else
56         currentState = -1;
57     }
58     else
59       currentState = -1;
60   }
61
62   return currentState == 0 && dataItr == dataLine.end();
63 }
```

Figure 2. `/usr/local/4301/src/lab04main.C` (Part 2 of 2)

```
newuser@csunix ~> cd 4301
newuser@csunix ~/4301> ./getlab.ksh 04
  * Checking to see if a folder exists for Lab 04. . .No
  * Creating a folder for Lab 04
  * Checking to see if Lab 04 has sample input and output files. . .Yes
  * Copying input and output files for Lab 04
    from folder /usr/local/4301/data/04 to folder ./04
  * Checking to see if /usr/local/4301/src/lab04main.C exists. . .Yes
  * Copying file /usr/local/4301/src/lab04main.C to folder ./04
  * Checking to see if /usr/local/4301/include/lab04.h exists. . .No
  * Copying file /usr/local/4301/src/Makefile to folder ./04
  * Adding a target of lab04 to targets2srcfiles
  * Touching file ./04/lab04.cpp
  * Edit file ./04/lab04.cpp in Notepad++
newuser@csunix ~/4301> cd 04
newuser@csunix ~/4301/04> ls
00.dat  00.out  Makefile  lab04.cpp  lab04main.C
newuser@csunix ~/4301/04> make lab04
g++ -g -Wall -std=c++11 -c lab04main.C -I/usr/local/4301/include -I.
g++ -g -Wall -std=c++11 -c lab04.cpp -I/usr/local/4301/include -I.
g++ -o lab04 lab04main.o lab04.o -L/usr/local/4301/lib -lm
newuser@csunix ~/4301/04> cat 00.dat
aaaaaa
bbbbbb
ccc
babac
caabb
bcbaa
abcba
ccaa
cabc
cccccc
aacbbccc
ccccccccc
aaaaaaaaaaaabbbbbbbbbbbb
aaabbb

a
b
aa
ab
ba
bb
aaa
bbb
abba
abcb
abbbbaa
```

Figure 3. Commands to Compile, Link, & Run Lab 04 (Part 1 of 3)

```
49 newuser@csunix ~/4301/04> cat 00.dat | ./lab04
50 Your Name
51 CS 4301
52 Lab 4
53 {x | x is in {a, b, c}+, and the number of a's plus the number of b's
54      plus twice the number of c's is divisible by six}
55
56 Input:  aaaaaa *** valid input ***
57
58 Input:  bbbbbb *** valid input ***
59
60 Input:  ccc *** valid input ***
61
62 Input:  babac *** valid input ***
63
64 Input:  caabb *** valid input ***
65
66 Input:  bcbaa *** valid input ***
67
68 Input:  abcba *** valid input ***
69
70 Input:  ccaa *** valid input ***
71
72 Input:  cabc *** valid input ***
73
74 Input:  cccccc *** valid input ***
75
76 Input:  aacbbccc *** valid input ***
77
78 Input:  ccccccccc *** valid input ***
79
80 Input:  aaaaaaaaaaaabbbbbbbbbbbb *** valid input ***
81
82 Input:  aaabbb *** valid input ***
83
84 Input:   --- invalid input ---
85
86 Input:  a --- invalid input ---
87
88 Input:  b --- invalid input ---
89
90 Input:  aa --- invalid input ---
91
92 Input:  ab --- invalid input ---
93
94 Input:  ba --- invalid input ---
95
96 Input:  bb --- invalid input ---
97
```

Figure 3. Commands to Compile, Link, & Run Lab 04 (Part 2 of 3)

```
98  Input:  aaa --- invalid input ---
99
100 Input:  bbb --- invalid input ---
101
102 Input:  abba --- invalid input ---
103
104 Input:  abcb --- invalid input ---
105
106 Input:  abbbbaa --- invalid input ---
107
108 newuser@csunix ~/4301/04> cat 00.dat | ./lab04 > my.out
109 newuser@csunix ~/4301/04> diff 00.out my.out
110 newuser@csunix ~/4301/04>
```

Figure 3. Commands to Compile, Link, & Run Lab 04 (Part 3 of 3)