C Supplementary

Terry Marris February 2013

Finite State Machines

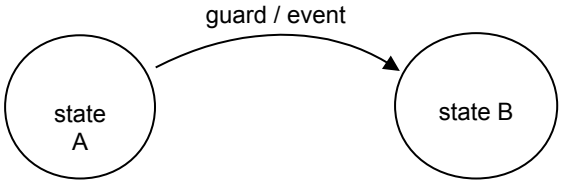
A finite state machine is modelled by a set of states, a function to get from one state to the next, and a function to produce output depending on the current state.

A program's state is represented by the values of its variables at a moment in time. For example, a program that maintains a file may have several states. In the *closed* state, the file is closed. In the *open-for reading* state data can be retrieved from the file. In the *open-for-writing* state data can be written to the file.

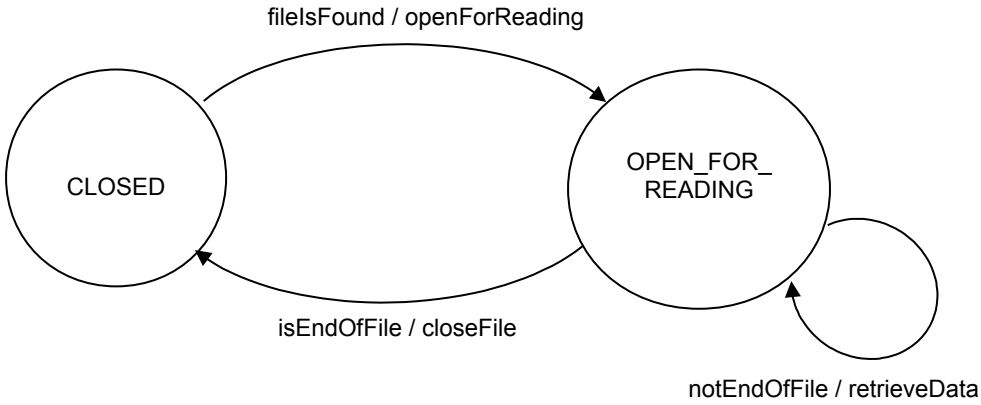
An event causes the state of an object to change. For example, a request to open the file might cause the state in a file-handling program to change from *closed* to *open for reading*.

State Transition Diagrams

A state transition diagram shows the relationships between states and events. In a state transition diagram, circles represent states, and arcs represent the transition from one state to the next.



A transition arc is labelled with the event that causes the state transition and, optionally, a guard or Boolean condition. For example, to print the contents of a file we might sketch



A state transition diagram describes the *behaviour* of the program.

State Transition Table

A state transition table shows the states of a system and the possible transitions from one state to another. A state transition table has four columns.

Current State	Event	Action	Next State
CLOSED	openTheFile	openForReading	OPEN_FOR_READING
OPEN_FOR_READING	retrieveRecords	retrieveRecord	OPEN_FOR_READING
OPEN_FOR_READING	endOfFileIsReached	closeFile	FINISHED
FINISHED		exit	

An *event* comes from outside the system and causes a transition from one state to the next. The next state depends on the current state as well as the event. An *action* is usually some function call that depends on the current state.

Record Retrieval

The program shown below retrieves and displays the contents of a text file. The text file is the source code itself. And the program structure used is based on the finite state machine model described above.

```
/* fsm.c: Finite State Machine example.
   Retrieves the contents of a text file.
*/

#include <stdio.h>

typedef enum { CLOSED, OPEN_FOR_READING, FINISHED } STATE;

char *fileName = "fsm.c";

int printLine(char *line)
{
  return printf("%s", line);
}


STATE nextState(STATE state)
{
  static FILE *file;
  char line[79];

  switch(state) {
  case CLOSED:
    file = fopen(fileName, "r");
    if (file != NULL)
      state = OPEN_FOR_READING;
    else
      state = FINISHED;
    break;

  case OPEN_FOR_READING:
    if (NULL != fgets(line, sizeof(line), file))
      printLine(line);
    else
      state = FINISHED;
    break;
```

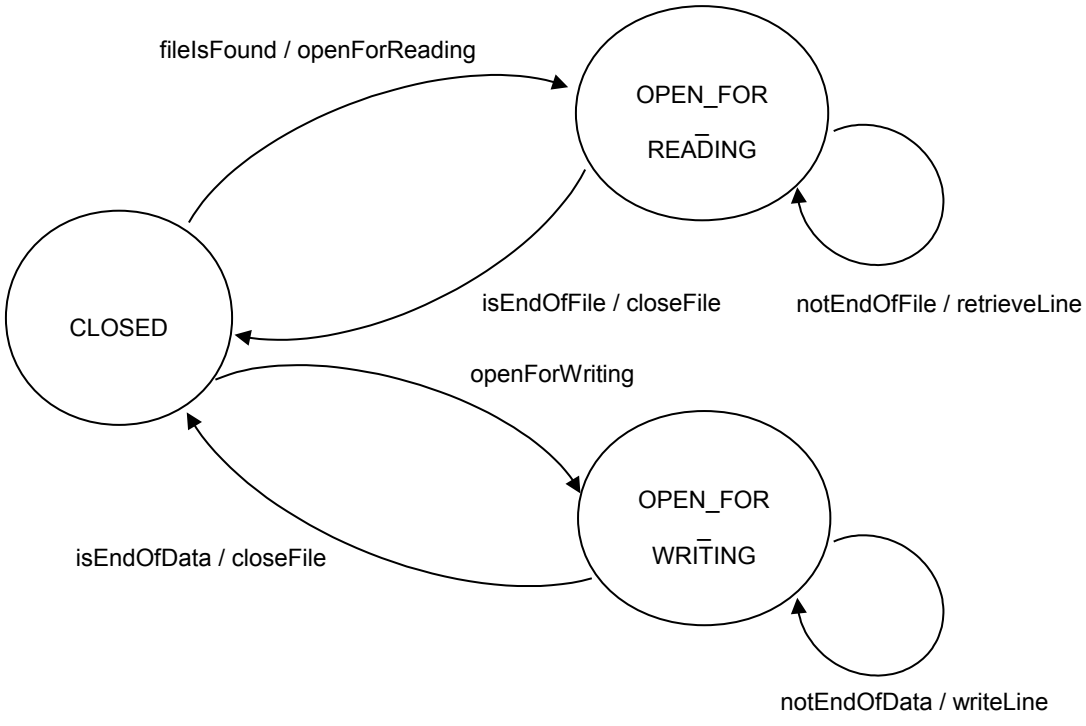
```
  case FINISHED:
    fclose(file);
    break;
  }
  return state;
}


int main()
{
  STATE state = CLOSED;

  while (state != FINISHED)
    state = nextState(state);
  return 0;
}
```

The file handling operations, *open*, *read* and *close*, are all contained within the same function, *nextState()*. *nextState()* is responsible for managing the transitions from one state to the next depending on the current state and events such as *open the file* and *end of file reached*.

The next program extends *fsm.c* shown above by providing functionality to create the file and then to display its contents.

File Input-Output

We start by sketching a state transition diagram to represent the program's intended behaviour, then go on to constructing the corresponding state transition table, before moving on to the coding.



Current State	Event	Action	Next State
CLOSED	READ	openForReading	OPEN_FOR_READING
OPEN_FOR_READING	retrieveRecords	retrieveLine	OPEN_FOR_READING
OPEN_FOR_READING	endOfFileIsReached	closeFile	FINISHED
CLOSED	WRITE	openForWriting	OPEN_FOR_WRITING
OPEN_FOR_WRITING	writeRecords	writeLine	OPEN_FOR_WRITING
OPEN_FOR_WRITING	endOfDataIsReached	closeFile	FINISHED
FINISHED		exit	

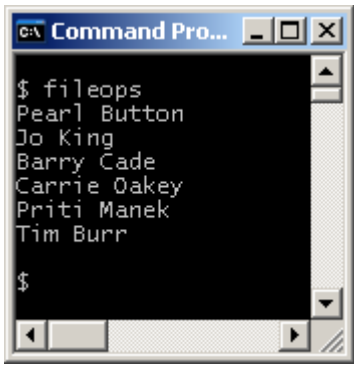
The program, and its run, are shown below.

```
/* fileops.c: Creates a text file,
              displays its contents
*/

#include <stdio.h>

typedef enum { CLOSED, OPEN_FOR_READING,
               OPEN_FOR_WRITING, FINISHED } STATE;

typedef enum { READ, WRITE } MODE;

char *fileName = "test.data";


int printLine(char *line)
{
  return printf("%s", line);
}


char *getLine()
{
  static int i = -1;
  static char *line[] = {
    "Pearl Button",
    "Jo King",
    "Barry Cade",
    "Carrie Oakey",
    "Priti Manek",
    "Tim Burr"
  };

  i++;
  if (i > 5)
    return NULL;
  return line[i];
}


STATE nextState(STATE state, MODE mode)
{
  static FILE *file;
  char line[79];
  char *str;
```

```
  switch(state) {
  case CLOSED:
    if (mode == READ) {
      file = fopen(fileName, "r");
      if (file != NULL)
        state = OPEN_FOR_READING;
      else
        state = FINISHED;
    }
    else if (mode == WRITE) {
      file = fopen(fileName, "w");
      if (file != NULL)
        state = OPEN_FOR_WRITING;
      else
        state = FINISHED;
    }
    break;

  case OPEN_FOR_READING:
    if (NULL != fgets(line, sizeof(line), file))
      printLine(line);
    else {
      fclose(file);
      state = FINISHED;
    }
    break;

  case OPEN_FOR_WRITING:
    str = getLine();
    if (str != NULL)
      fprintf(file, "%s\n", str);
    else {
      fclose(file);
      state = FINISHED;
    }
    break;

  case FINISHED:
    break;
  }
  return state;
}


int writeToFile()
{
  STATE state = CLOSED;

  while (state != FINISHED)
    state = nextState(state, WRITE);
  return 0;
}

int readFromFile()
{
  STATE state = CLOSED;

  while (state != FINISHED)
    state = nextState(state, READ);
  return 0;
}
```

```
int main()
{
  writeToFile();
  readFromFile();
  return 0;
}
```

The program run is shown below.



Well, what do you think? Personally, I love the fact that program control is confined to just the one function, as are all the file I/O operations.

Exercise

1 Extend the program *fileops.c* shown above so that it appends lines of text to an existing file. The program should create the file if it does not exist.

Bibliography

JACKY J The Way of Z Cambridge University Press 1997
MARCH D.L. *www4.desales.edu/~dlm1/it532/class09/statetab.html* accessed Sep 2007.