Finite State Machines

CIS 15 : Spring 2007

Functionalia

HW3 is OUT, due Sunday, March 25th, 11:59 pm

Midterms and HW's returned after the weekend.

Today:

- XML Basics
- Finite State Machines
- Begin Memory Organization

XML Basics

XML (e**X**tensible **M**arkup **L**anguage) - describes structured data.

SVG (**S**calable **V**ector **G**raphics) - is a special type of data.

Tags

- Describe Basic Elements:

```
<rect />
```

- Contain Data:

```
<desc> This is my description! </desc>
```

Attributes

- Parameters associated to the tag (and it's data):

```
<rect x="4" y="10" width="20" height="30" />
```

Robustness of XML

Whitespace Tolerant

- Any amount of whitespace can exists between elements and attributes

```
<rect x="4" y="10"
                              width="20" height="30"
/>
```

Default Behavior

```
<rect x="20" />
```

- **y and any other required attributes have default values associated to it**

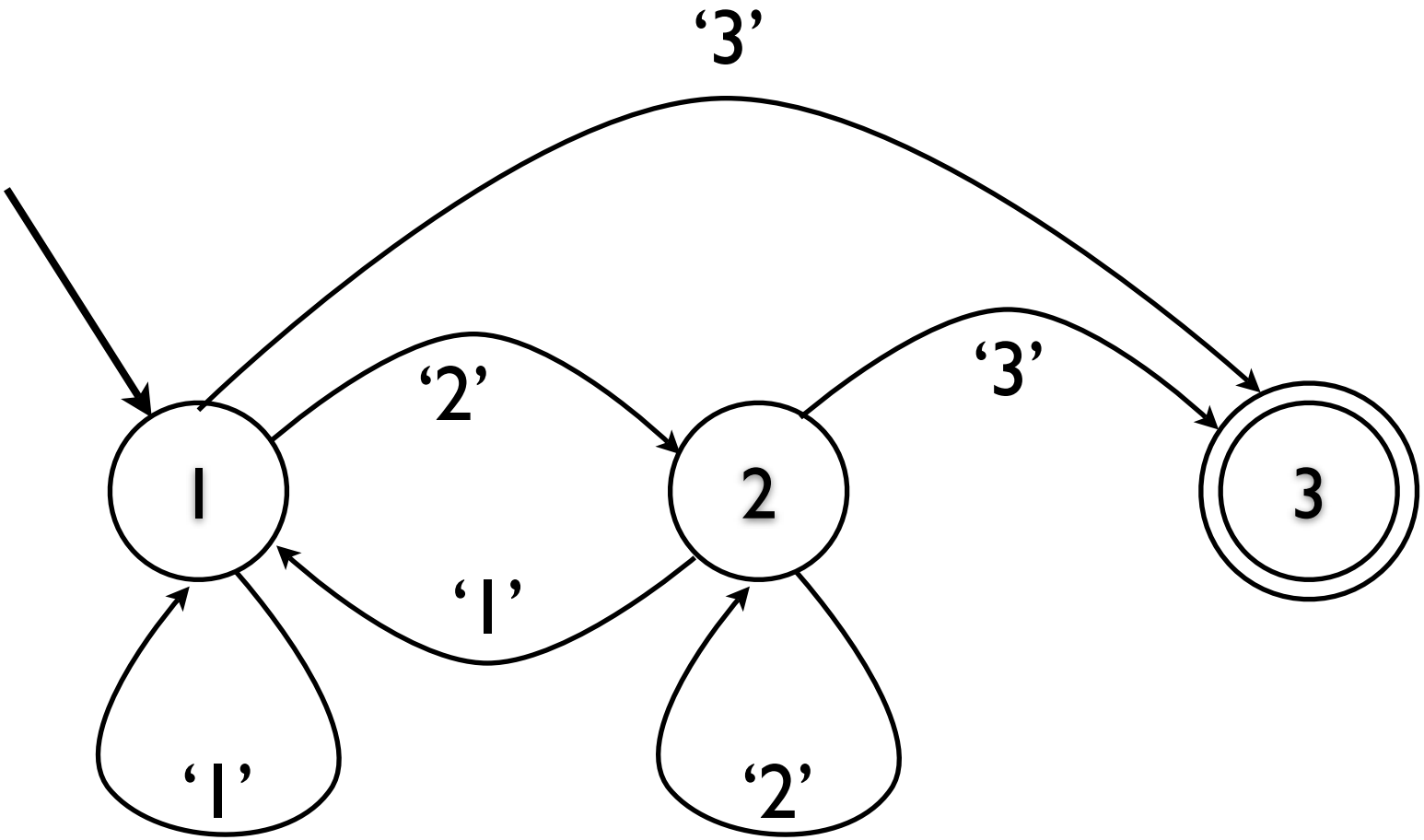
Error? Ignore it!

```
<rct />
```

Finite State Machine

Finite State Machine is a computational structure that encodes states and transition between states based on some input or action.

Symbolized by a *State Diagram* - which is a *Directed Graph* structure.

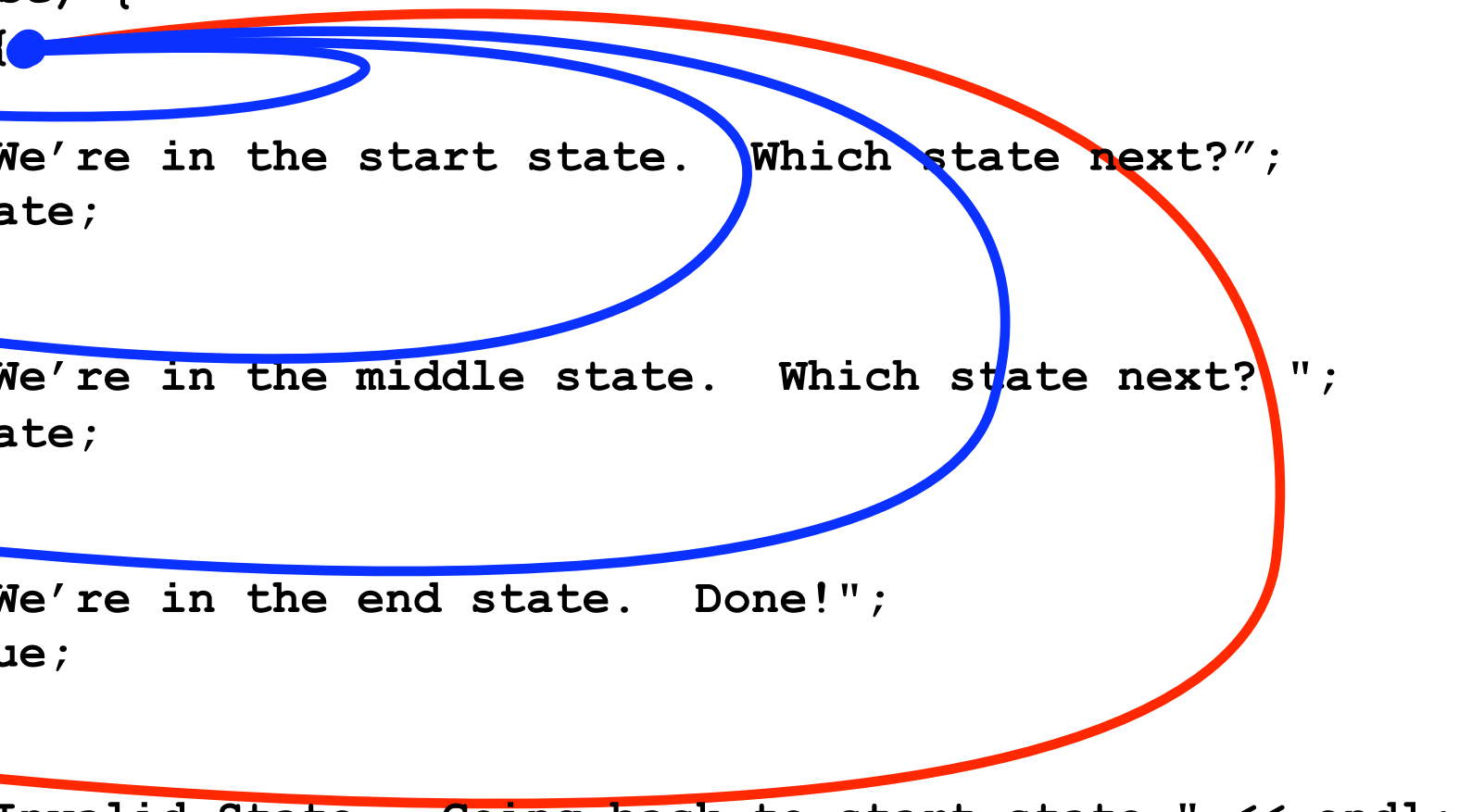


Using a `switch()` to represent a Finite State Machine

```
bool quit = false;
int state = 1;

while(quit == false) {
   switch(state) {
      case 1: // Start State
         cout << "We're in the start state.  Which state next?";
         cin >> state;
      break;
      case 2:
         cout << "We're in the middle state.  Which state next? ";
         cin >> state;
      break;
      case 3:
         cout << "We're in the end state.  Done!";
         quit = true;
      break;
      default:
         cout << "Invalid State.  Going back to start state." << endl;
         state = 1;
         break;
      }
}
```

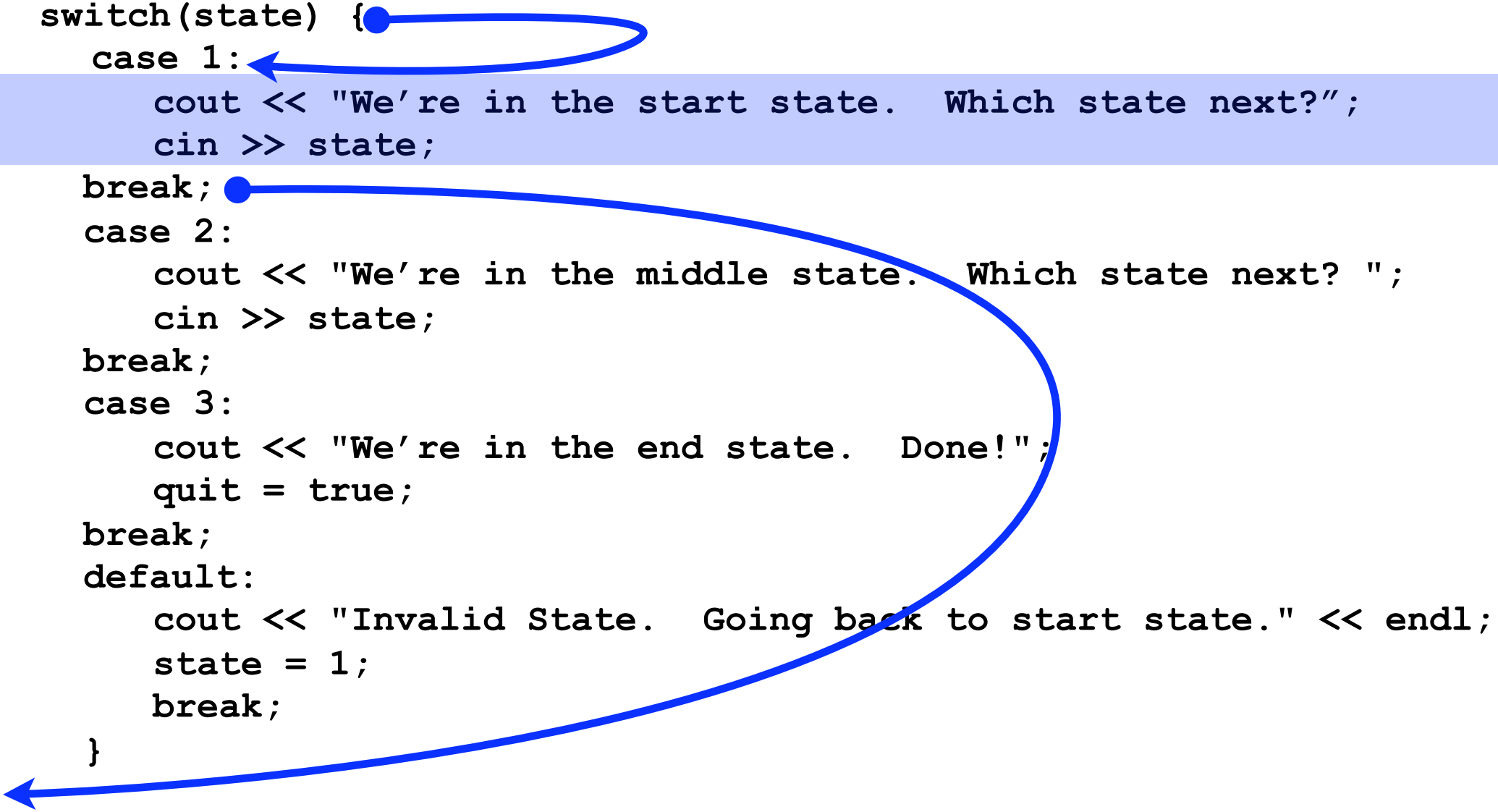
How does a `switch()` work?

```
bool quit = false;
int state = 1;

while(quit == false) {
   switch(state) {
      case 1:
         cout << "We're in the start state.  Which state next?";
         cin >> state;
      break;
      case 2:
         cout << "We're in the middle state.  Which state next? ";
         cin >> state;
      break;
      case 3:
         cout << "We're in the end state.  Done!";
         quit = true;
      break;
      default:
         cout << "Invalid State.  Going back to start state." << endl;
         state = 1;
         break;
      }
}
```

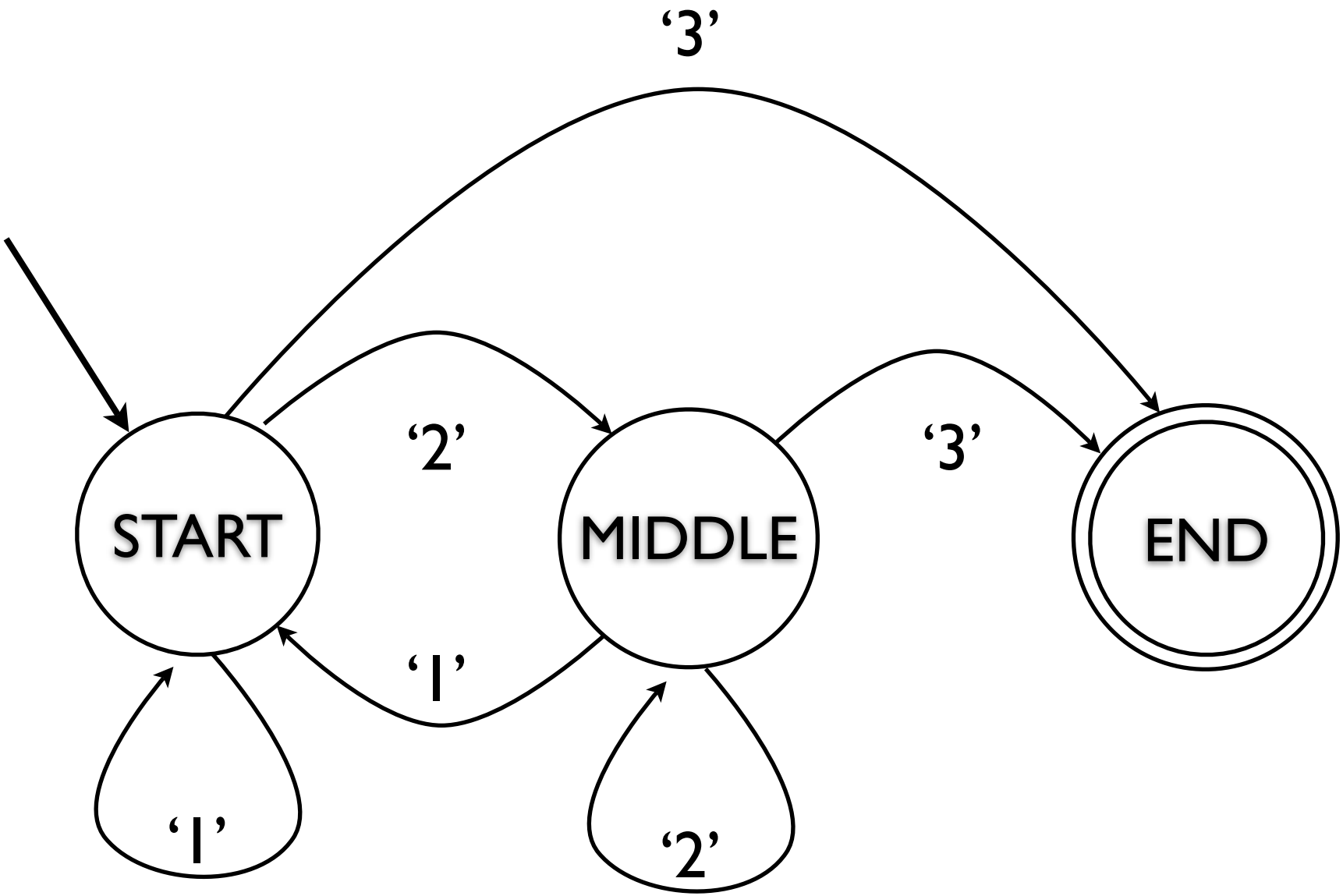


How does a `switch()` work?

```
bool quit = false;
int state = 1;

while(quit == false) {
   switch(state) {
      case 1:
         cout << "We're in the start state.  Which state next?";
         cin >> state;
      break;
      case 2:
         cout << "We're in the middle state.  Which state next? ";
         cin >> state;
      break;
      case 3:
         cout << "We're in the end state.  Done!";
         quit = true;
      break;
      default:
         cout << "Invalid State.  Going back to start state." << endl;
         state = 1;
         break;
      }
}
```



Give the States Names



Defines and Enums as States

```
#define START 1
#define MIDDLE 2
#define END 3
```

```
enum {START, MIDDLE, END};
```

Just to be Sure, start your Enums with a value

```
#define START 1
#define MIDDLE 2
#define END 3
```

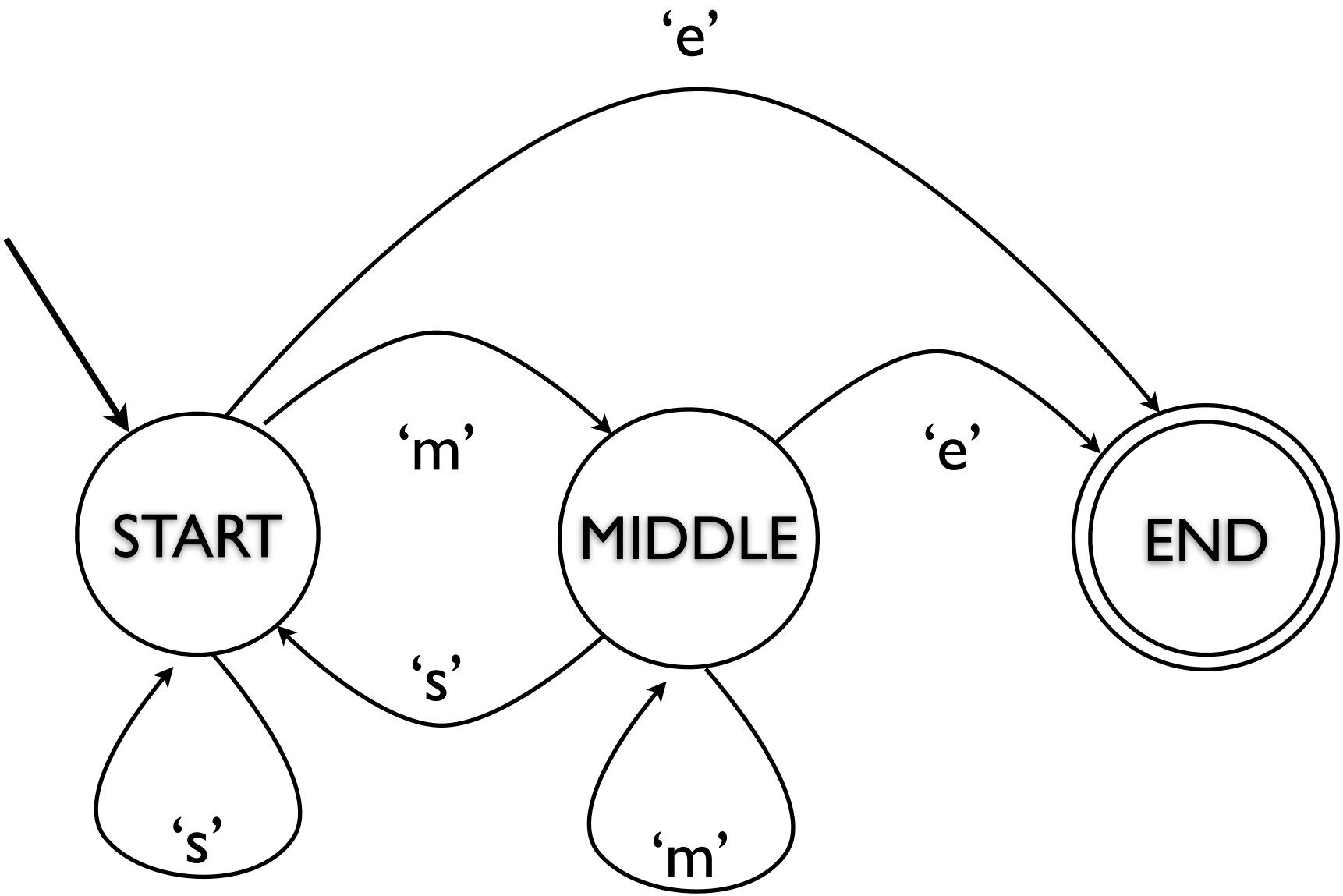
```
enum {START, MIDDLE, END};
```

```
enum {START = 1, MIDDLE, END};
```

Using a `switch()` to represent a Finite State Machine

```
bool quit = false;
enum {START = 1, MIDDLE, END};
int state = START;

while(quit == false) {
   switch(state) {
      case START:
         cout << "We're in the start state.  Which state next?";
         cin >> state;
      break;
      case MIDDLE:
         cout << "We're in the middle state.  Which state next? ";
         cin >> state;
      break;
      case END:
         cout << "We're in the end state.  Done!";
         quit = true;
      break;
      default:
         cout << "Invalid State.  Going back to start state." << endl;
         state = START;
         break;
      }
}
```

Have Different Input Cause Transitions



Write a Function that gets the Next State

```
enum {START = 1, MIDDLE, END, UNDEFINED};

int getNextState()
{
      char theInput;
      cin >> theInput;

      if(theInput == 's')
      {
          return START;
      }
      else if(theInput == 'm')
      {
          return MIDDLE;
      }
      else if(theInput == 'e')
      {
          return END;
      }
      else
      {
          return UNDEFINED;
      }
}
```

Using `getNextState()` to Map Input to State Transition

```
bool quit = false;
enum {START = 1, MIDDLE, END, UNDEFINED};
int state = START;
...
while(quit == false) {
   switch(state) {
      case START:
         cout << "We're in the start state.  Which state next?";
         state = getNextState();
      break;
      case MIDDLE:
         cout << "We're in the middle state.  Which state next? ";
         state = getNextState();
      break;
      case END:
         cout << "We're in the end state.  Done!";
         quit = true;
      break;
      default:
         cout << "Invalid State.  Going back to start state." << endl;
         state = START;
         break;
      }
}
```

Using `getNextState()` to Map Input to State Transition

```
bool quit = false;
enum {START = 1, MIDDLE, END, UNDEFINED};
int state = START;
...
while(quit == false) {
   switch(state) {
      case START:
         cout << "We're in the start state.  Which state next?";
         state = getNextState();
      break;
      case MIDDLE:
         cout << "We're in the middle state.  Which state next? ";
         state = getNextState();
      break;
      case END:
         cout << "We're in the end state.  Done!";
         quit = true;
      break;
      default:
         cout << "Invalid State.  Going back to start state." << endl;
         state = START;
         break;
      }
}
```

States are now **independent** from their integer Representation

Using `getNextState()` to Map Input to State Transition

```
bool quit = false;
enum {START, MIDDLE, END, UNDEFINED};
int state = START;
...
while(quit == false) {
   switch(state) {
      case START:
         cout << "We're in the start state.  Which state next?";
         state = getNextState();
      break;
      case MIDDLE:
         cout << "We're in the middle state.  Which state next? ";
         state = getNextState();
      break;
      case END:
         cout << "We're in the end state.  Done!";
         quit = true;
      break;
      default:
         cout << "Invalid State.  Going back to start state." << endl;
         state = START;
         break;
      }
}
```

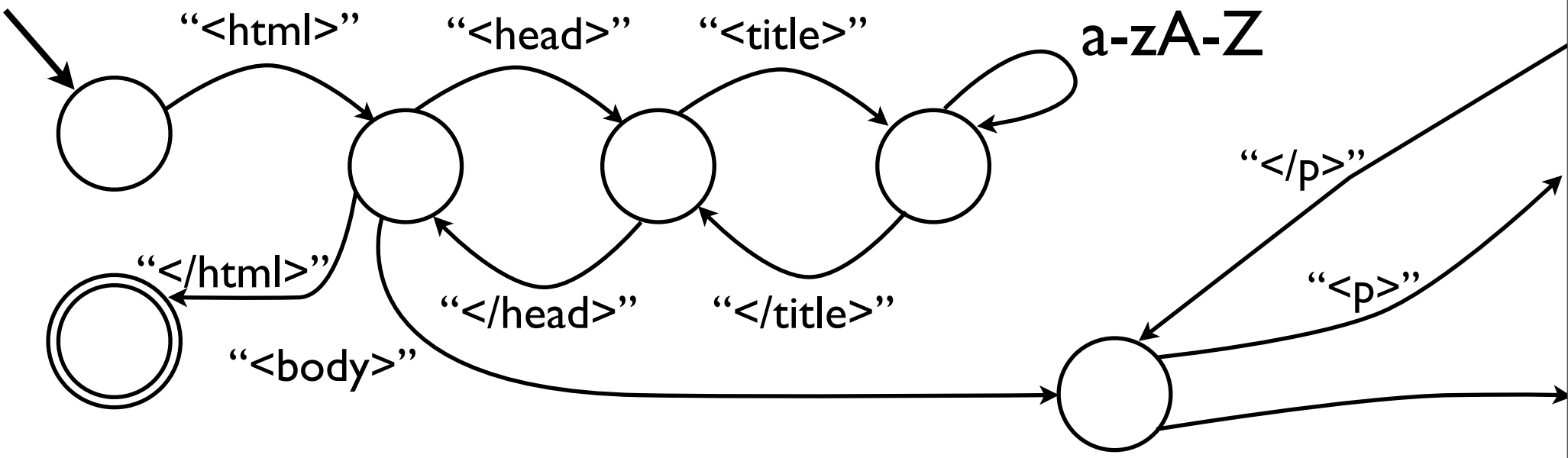
States are now **independent** from their integer Representation

Use Finite State Machines to Help in Parsing Files

```
<html>
   <head>
      <title> Parse this </title>
   </head>
   <body>
      <p> Parsing html pages is <b>fresh</b>! </p>
   </body>
</html>
```

Use Finite State Machines to Help in Parsing Files

```
<html>
   <head>
      <title> Parse this </title>
   </head>
   <body>
      <p> Parsing html pages is <b>fresh</b>! </p>
   </body>
</html>
```



Draw a FSM that parses this XML structure.

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"

  "http://www.w3.org/TR/2002/WD-SVG11-20020215/DTD/svg11.dtd">

<svg width="12cm" height="4cm" viewBox="0 0 1200 400"

     xmlns="http://www.w3.org/2000/svg" version="1.1">

  <desc>Rectangular Test Sample</desc>


  <!-- Outlined Rectangle -->

  <rect x="1" y="1" width="400" height="400"

        fill="none" stroke="blue" stroke-width="2"/>


  <!-- One Thick Line in Rectangle -->

  <line x1="100" y1="300" x2="300" y2="100" stroke="red" stroke-
width="10"  />

</svg>
```