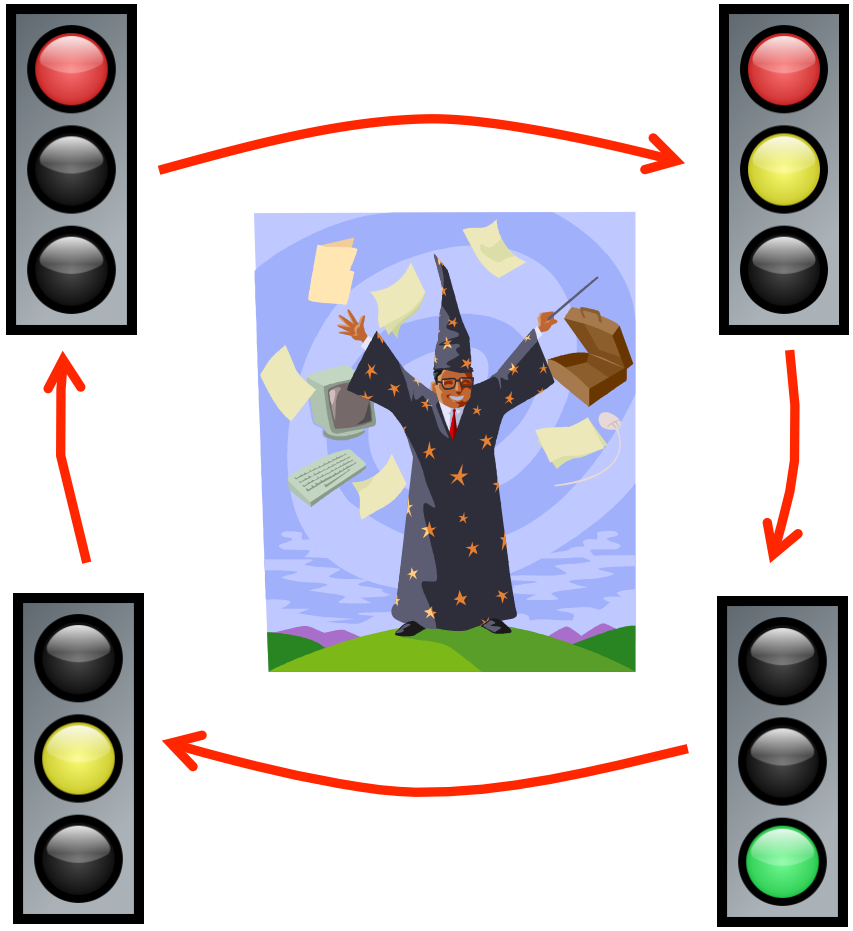
Multi-state systems



Overview

- Avoiding magic numbers
  - Variables takes on a small set of values
  - Use descriptive names instead of literal values
- Testing a variable against many values
  - Execute a block of code depending on the value
  - Avoid a big if-else-if-else if-… block
- Creating a multi-state system
  - Stop light
  - Stop light v2.0

Variables from a set of values

- Magic numbers
  - Where did the value come from?
  - What does it mean?
  - What if you mistype the number?
  - What if you want to keep value in specific range?


```
char direction = 0;

...

if ((direction == 1) || (direction == 3) ||
    (direction == 5) || (direction == 7))
{ /* TBD */ }

direction = 8;       // Valid???
direction = -27;     // Valid???
```

- Solution 1: Use #define preprocessor
  - Descriptive names means everybody can read
  - Bugs less likely, typo in name = compile error

```
#define NORTH      0
#define NORTHEAST  1
#define EAST       2
#define SOUTHEAST  3
#define SOUTH      4
#define SOUTHWEST  5
#define WEST       6
#define NORTHWEST  7

void main()
{
   char direction = NORTH;

   ...

   if ((direction == NORTHEAST) || (direction == SOUTHEAST) ||
       (direction == SOUTHWEST) || (direction == NORTHWEST))
   { /* TBD */ }
```

#define not always ideal

```
#define NORTH       0
#define NORTHEAST   1
#define EAST        2
#define SOUTHEAST   3
#define SOUTH       4
#define SOUTHWEST   5
#define WEST        6
#define NORTHWEST   7

void main()
{
   char direction = NORTH;

   ...

   if ((direction == NORTHEAST) || (direction == SOUTHEAST) ||
       (direction == SOUTHWEST) || (direction == NORTHWEST))
   { /* TBD */ }

direction = 8;     // Valid???
direction = -27;   // Valid???
```

Problem 1: Tedious to type. Also easy to mess up, e.g. setting two constants to same value.

Problem 2: No warning if we don't use a friendly name.

Enumerations

- A better solution: enumerations
  - Specifies exact set of friendly names

```
typedef enum {NORTH, NORTHEAST, EAST, SOUTHEAST,
              SOUTH, SOUTHWEST, WEST, NORTHWEST} Compass;

void main()
{
   Compass direction = NORTH;

   if ((direction == NORTHEAST) ||
       (direction == SOUTHEAST) ||
       (direction == SOUTHWEST) ||
       (direction == NORTHWEST))
   { /* TBD */ }

    direction = 8;

   ...
```

Easier to declare that a list of #define's

Now a compiler warning.
Way to watch our back compiler!

Conditional action from a set

- Do something depending on a value value
  - if-else if-else if... statements can get tedious

```
if (day == 1)
   displayLCD("Monday");
else if (day == 2)
   displayLCD("Tuesday");
else if (day == 3)
   displayLCD("Wednesday");
else if (day == 4)
   displayLCD("Thursday");
else if (day == 5)
   displayLCD("Friday");
else if (day == 6)
   displayLCD("Saturday");
else if (day == 7)
   displayLCD("Sunday");
else
   displayLCD("Invalid day!");
```

Display a different string to an LCD depending on the day of the week.

Conditional action from a set

- switch statement
  - Works integer types and enumerations

```
switch (day)
{
   case 1:  displayLCD("Monday");       break;
   case 2:  displayLCD("Tuesday);       break;
   case 3:  displayLCD("Wednesday");    break;
   case 4:  displayLCD("Thursday");     break;
   case 5:  displayLCD("Friday");       break;
   case 6:  displayLCD("Saturday");     break;
   case 7:  displayLCD("Sunday");       break;
  default: displayLCD("Invalid day!"); break;
}
```

case block normally ends with a break

default block is optional, but if present executes if no other case matched. Like the else in an if-else if-else statement.

Buggy switch statement

```
Compass direction = NORTH;

switch (direction)
{
   case NORTH:
      y++;
      displayLCD("Walking north");
   case SOUTH:
      y--;
      displayLCD("Walking south");
   case EAST:
      x++;
      displayLCD("Walking east");
   case WEST:
      x--;
      displayLCD("Walking west");
}
```

case blocks with fall through to next block if you don't use the break statement!

```
Output:

Walking north
Walking south
Walking east
Walking west
```

Falling through cases

```
Compass direction = SOUTHEAST;

switch (direction)
{
   case NORTHWEST:
   case NORTHEAST:
   case NORTH:
      displayLCD("Heading northbound");
      break;
   case SOUTHWEST:
   case SOUTHEAST:
   case SOUTH:
      displayLCD("Heading southbound");
      break;
}
```

Sometimes falling through to next case block is what you want.

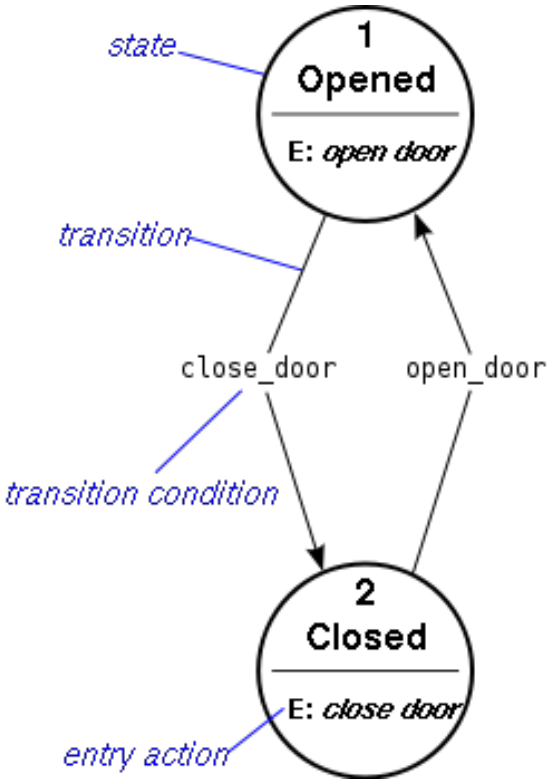
Easy way to do same thing for a set of discrete values.

```
Output:

Heading southbound
```

Multi-state systems

- Model system that transitions through a series of different states
  - Used for many embedded applications
  - Each state may:
    - Read input
    - Generate output
    - Wait for certain amount of time to pass
    - Call functions
    - Transition to another state

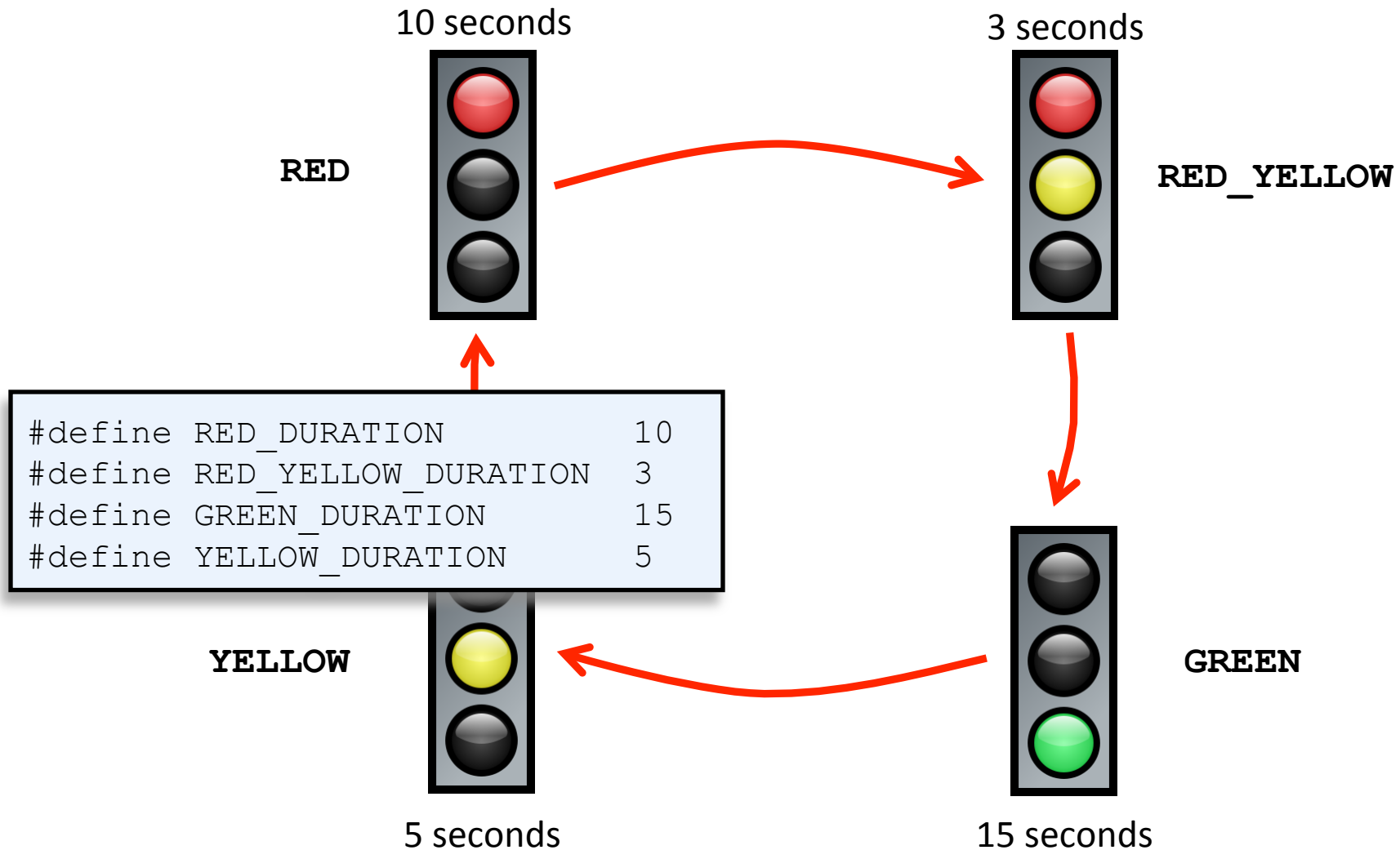


Categories of multi-state systems

- Timed
  - Transition depends entirely on time
  - Execute A for 10 seconds, then B for 5 seconds, …
  - e.g. Simple traffic light
- Input/Timed
  - Uses input and/or time to transition
  - Transition from B back to A if no input for 5 seconds
  - e.g. Microwave, autopilot, security system
- Input
  - No concept of time, uses only input to transition
  - e.g. Simple thermostat, calculator

Traffic light state machine

```
typedef enum {RED, RED_YELLOW, GREEN, YELLOW} LightState;
```

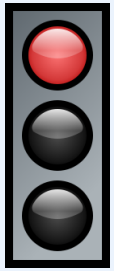


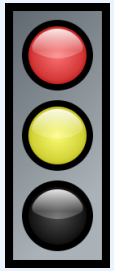
Basic state machine, 1/2

```
unsigned char sleepCount  = 0;
unsigned char secsInState = 0;
LightState state = RED;

startTimer0();  // Start timer0 0.01s heartbeat

while (1)
{
   switch (state)
   {
      case RED:
      {
         if (secsInState >= RED_DURATION)
         {
            state = RED_YELLOW;
            secsInState = 0;
         }
         break;
      }
      case RED_YELLOW:
      {
         if (secsInState >= RED_YELLOW_DURATION)
         {
            state = GREEN;
            secsInState = 0;
         }
         break;
      }
...
```

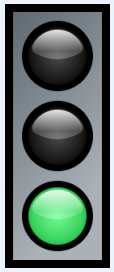


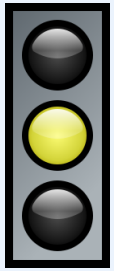


Basic state machine, 2/2

```
      case GREEN:
      {
         if (secsInState >= GREEN_DURATION)
         {
            state = YELLOW;
            secsInState = 0;
         }
         break;
      }
      case YELLOW:
      {
         if (secsInState >= YELLOW_DURATION)
         {
            state = RED;
            secsInState = 0;
         }
         break;
      }
   }

   sleep();
   sleepCount++;
   if (sleepCount > 99)
   {
      sleepCount = 0;
      secsInState++;
   }
}
```



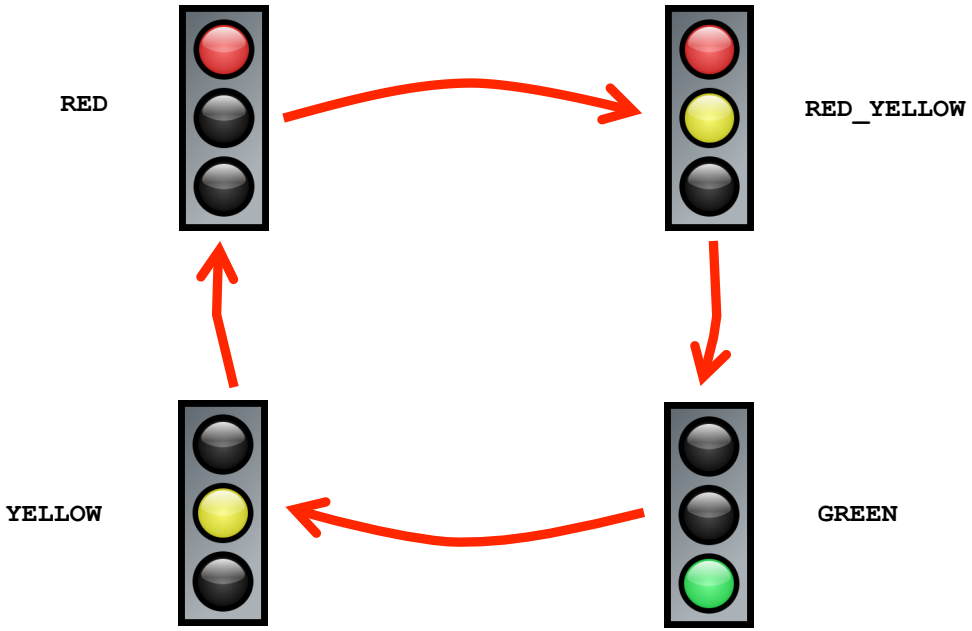


Responding to input

- New requirement:
  - Emergency vehicles transmit radio signal
  - Transition to red light going through yellow
    - If already in yellow, finish yellow cycle and go to red
    - If already in red, stay in red
  - Once in red, stays red for 20 seconds then back to normal
    - If another signal received while in red, extend for 20 seconds from time of signal 1-0 transition
  - New white light, lit when emergency mode active

Reading the input

- Radio detector on P2.0
  - P2.0 low when signal detected
  - Signal must be detected continuously for >= 0.05s
    - Otherwise consider it random detector noise
  - Event triggered on 0-1 transition on P2.0
  - Function normally while waiting for 0-1 transition of signal

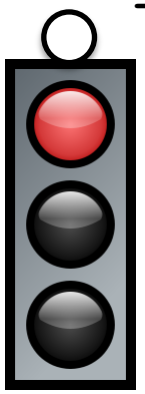


Stop Light v2.0

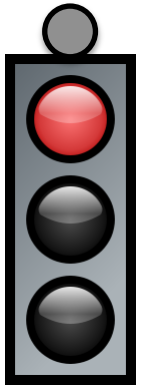
- Fill in transitions on state diagram
  - Label transitions with what causes transition
- Fill in the missing code:
  - checkSignal() function
  - State transitions on an emergency signal
  - Code for the emergency yellow/red states
  - Use friendly names for constants (see top of source)

Stop Light v2.0 states

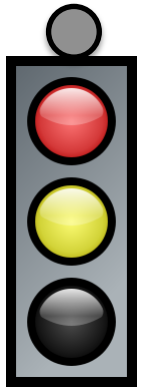
EMERGENCY_RED

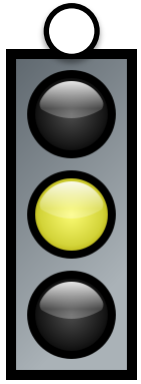


RED

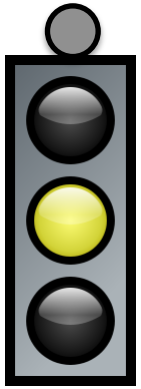


RED_YELLOW

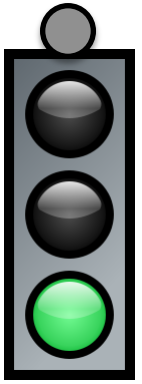




EMERGENCY_YELLOW



YELLOW



GREEN