Solutions to Assignment 7
State Machines in C

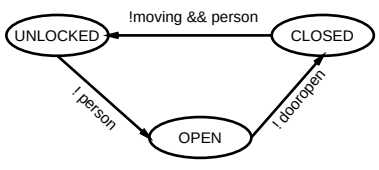
Question 1

Using the function names given in the question the inputs are: `moving` (bus moving), `person` (person waiting to exit), and `dooropen` (door open). The outputs are: `lock`, and `brakes`.

The specification requires 3 combinations of outputs: (1) the brakes off and the door locked while the bus is moving, (2) the brakes on and the door unlocked while someone is waiting to get off, and (3) the brakes on and the door locked while the passenger is getting off. The controller must therefore have at least 3 states. We will assign these states the labels CLOSED, UNLOCKED, and OPEN, respectively. We will attempt to produce a correct design with this many states and add additional states if required to meet the specifications. The outputs for each state are thus:

state	`lock`	`brakes`
CLOSED	1	0
UNLOCKED	0	1
OPEN	1	1

A simple state transition diagram would be as follows:



The state transition table is as follows:

current state	input conditions			next state
	`moving`	`person`	`door open`	
CLOSED	0	0	X	CLOSED
CLOSED	0	1	X	UNLOCKED
CLOSED	1	0	X	CLOSED
CLOSED	1	1	X	CLOSED
UNLOCKED	X	0	X	OPEN
UNLOCKED	X	1	X	UNLOCKED
OPEN	X	X	0	CLOSED
OPEN	X	X	1	OPEN

A C program to implement this state machine would be as follows:

```
/*
   APSC 380 Assignment 7 Question 1
   Ed Casas, 1997/10/05
*/

#define CLOSED    1
#define UNLOCKED  2
#define OPEN      3

main()
{
  int state = CLOSED ;

  while ( 1 ) {

    /* read input and set next state */

    if ( state == CLOSED ) {

      lock ( 1 ) ;
      brakes ( 0 ) ;

      if ( !moving() && person() ) {
        state = UNLOCKED ;
      }

    } else if ( state == UNLOCKED ) {

      lock ( 0 ) ;
      brakes ( 1 ) ;

      if ( ! person() ) {
        state = OPEN ;
      }
```

```
    } else if ( state == OPEN ) {

      lock ( 1 ) ;
      brakes ( 1 ) ;

      if ( !dooropen() ) {
        state = CLOSED ;
      }

    }
  }

}
```

Question 2

The input is the character returned by the `getche()` function and the output is the character output using the `putchar()` function.

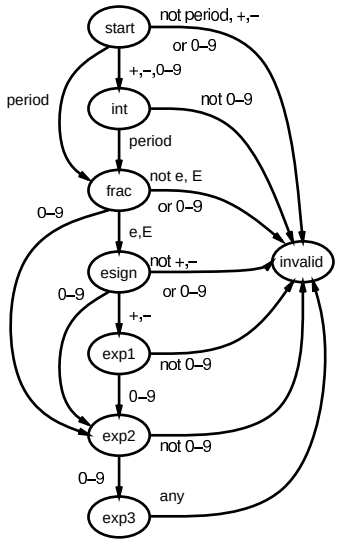
Although in this case there are only two possible outputs (`y` and `n`), the state machine needs more than two states. This is because allowed input characters depend on the characters that have already been input. For example, the letter 'E' is only valid when starting the exponent part of the number. The solution below defines seven different "valid input" states for the different sets of allowed input characters. The EXP3 state is used so that the output immediately after the second digit of the exponent is input can remain `y`. The state names and their meanings are as follows:

State	Character Expected
START	initial character
INT	integer digits
FRAC	fractional digits
ESIGN	sign of exponent
EXP1	first digit of exponent
EXP2	second digit of exponent
EXP3	character after exponent
INVALID	no more valid characters allowed

The outputs for each state are as follows:

State	Output
START	y
INT	y
FRAC	y
ESIGN	y
EXP1	y
EXP2	y
EXP3	y
INVALID	n

The state transition diagram is as follows:



The state transition table is:

current state	input conditions `getche`	next state
START	+,-,0-9	INT
START	.	FRAC
START	not +,-,0-9,.	INVALID
INT	0-9	INT
INT	.	FRAC
INT	not .,0-9	INVALID
FRAC	0-9	FRAC
FRAC	E,e	ESIGN
FRAC	not E,e,0-9	INVALID
ESIGN	+,-	EXP1
ESIGN	0-9	EXP2
ESIGN	not +,-,0-9	INVALID
EXP1	0-9	EXP2
EXP1	not 0-9	EXP3
EXP2	0-9	EXP3
EXP2	not 0-9	INVALID
EXP3	any	INVALID
INVALID	any	INVALID

A possible C program is:

```
/*
   APSC 380 Assignment 7 Question 2
   Ed Casas, 1997/10/05
*/

#include <stdio.h>

#define getche() getc(stdin)

#define START   1
#define INT     2
#define FRAC    3
#define ESIGN   4
#define EXP1    5
#define EXP2    6
#define EXP3    7
#define INVALID 8

main()
{
  int cin, cout, state = START ;

  while ( 1 ) {

    /* read input and set next state */

    cin = getche() ;

    if ( state == START ) {

      /* no characters read yet */

      if ( cin == '+' || cin == '-' || isdigit(cin) ) {
        state = INT ;
      } else if ( cin == '.' ) {
        state = FRAC ;
      } else {
        state = INVALID ;
      }

    } else if ( state == INT ) {

      /* expecting digits for integer part */

      if ( isdigit(cin) ) {
        state = INT ;
      } else if ( cin == '.' ) {
        state = FRAC ;
      } else {
        state = INVALID ;
      }

    } else if ( state == FRAC ) {

      /* expecting fraction digits  */

      if ( isdigit(cin) ) {
        state = FRAC ;
      } else if ( cin == 'E' || cin == 'e' ) {
        state = ESIGN ;
      } else {
        state = INVALID ;
      }

    } else if ( state == ESIGN ) {

      /* expecting exponent sign or first digit */

      if ( cin == '+' || cin == '-' ) {
        state = EXP1 ;
      } else if ( isdigit(cin) ) {
        state = EXP2 ;
      } else {
        state = INVALID ;
      }

    } else if ( state == EXP1 ) {

      /* expecting first exponent digit */

      if ( isdigit(cin) ) {
        state = EXP2 ;
      } else {
        state = INVALID ;
      }

    } else if ( state == EXP2 ) {

      /* expecting second exponent digit */

      if ( isdigit(cin) ) {
        state = EXP3 ;
      } else {
        state = INVALID ;
      }

    } else if ( state == EXP3 ) {

      /* no more valid characters possible */

      state = INVALID ;

    } else if ( state == INVALID ) {

      /* at least one invalid character seen */

    }


    /* output character depending on state */

    if ( state == INVALID ) {
      putchar ( 'n' ) ;
    } else {
      putchar ( 'y' ) ;
    }

  }

  printf ( "\n" ) ;

}
```