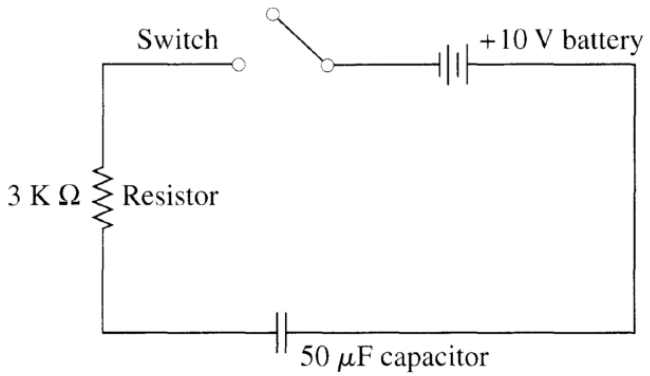
Project 2, Program Design

1. A professor will assign homework today and does not want it due on anybody's holy day. The professor enters today's day of the week (0 for Sunday, 1 for Monday, etc.) and the number of days, D, to allow the students to do the work, which may be several weeks. Calculate the day of the week on which the work would be due. If that day is someone's holy day – Friday (Moslems), Saturday (Jews), or Sunday (Christians) – add enough days to D to reach the following Monday. Print the corrected value of D and the day of the week the work is due.

    1) Use a switch statement for calculating the number of days based on today's day.
    2) If the entered day for today is not in the range of 0 to 6, display an error message and abort the program.
    3) Display the corrected value of the number of days and the day of the week the work is due. For example, if today is Thursday, and the number of days D is 8, then the due date falls on Friday, then 3 days will be added to reach the following Monday. So the corrected value of D is 11.

2. Write a C program that will calculate the voltage across the capacitor in the following circuit as time increases from 0 to 1 second in increments of 1/15 second, assuming that the switch is closed at time 0.



The voltage across the capacitor at time t is given by the following equation:

$$v(t) = V \times \left(1 - e^{-\frac{t}{\tau}}\right)$$

Constants: In this circuit, V = 10 volts, R = 3,000 ohms, and C = 50 x 10^{-6} farads. The circuit has a time constant τ, which depends on the resistance, R, and the capacitance, C, as τ = R x C = 0.15 second.

1. Use a for loop.
2. Use the math library function **exp(x)** to compute e^x. You will need to include the system header file math.h.
3. On Unix you will need **–lm** in your command line to tell the Linker to search the math library.
4. Use macro definition for all the constants.
5. Format the output so the output looks like the following. The time and voltage should display two digits after the decimal point.

```
time (sec)    voltage
  0.00            0.00
  0.07            3.59
  0.13            5.89
  0.20            7.36
  0.27            8.31
  0.33            8.92
  0.40            9.31
  0.47            9.55
  0.53            9.71
  0.60            9.82
  0.67            9.88
  0.73            9.92
  0.80            9.95
  0.87            9.97
  0.93            9.98
  1.00            9.99
```

Before you submit

1. Compile both programs with –Wall. –Wall shows the warnings by the compiler. Be sure it compiles on ***circe*** with no errors and no warnings.

   gcc –Wall holy_day.c

   gcc –Wall voltage.c

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

   chmod 600 holy_day.c

   chmod 600 voltage.c

3. Test the first program with the shell script try_holy_day on Unix:

   chmod +x try_holy_day

   ./try_holy_day

4. Submit holy_day. c and voltage.c on Canvas.

Grading

Total points: 100 (50 point each program)

1. A program that does not compile will result in a zero for that program.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80%

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **name**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
4. Use consistent indentation to emphasize block structure.

5. Full line comments inside function bodies should conform to the indentation of the code where they appear.
6. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **`#define PI 3.141592`**
7. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
8. Use either underscores or capitalization for compound names for variable: **`tot_vol`**, **`total_volumn`**, or **`totalVolumn`**.