Project 1, Program Design

1. Write a C program that converts knuts to sickles and galleons (the currency of the Harry Potter novels). The user will enter the total number of knuts. There are 29 knuts in one sickle and 17 sickles in one galleon.

Hint: Divide the amount by the number of knuts in one galleon to determine the number galleons, and then reduce the amount by the total value of the galleons (You can also use a remainder operator). Repeat for sickles. Be sure to use integer values throughout, no floating-point numbers.

If the amount entered is less than zero or greater than one billion (1000000000), output an error message and abort the program.

2. 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.

Before you submit:

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

   ```
   gcc –Wall knuts.c

   gcc –Wall due_date.c
   ```

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

   ```
   chmod 600 knuts.c
   chmod 600 due_date.c
   ```

3. Test your programs with the shell scripts on Unix:

   ```
   chmod +x try_knuts
   ./try_knuts

   chmod +x try_due_date
   ./try_due_date
   ```

4. Download *knuts.c* and *due_date.c* from the student cluster and submit on Canvas.

Grading:

Total points: 100 (50 points each program)

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