Project 1, Program Design

1. Tampa international airport offers long term parking at the following rates:

First 60 minutes is free;

61-80 minutes $4;

Each additional 20 minutes $2;

And $18 max. per day (24 hours).

Write a program *longterm_parking.c* that calculates and prints the charges for parking at the long term parking garage.

1. The user enters the number of total hours and minutes; the program prints the charge.
2. If the input is invalid, print a message and exit the program.

Example input/output:

```
Enter hours parked: 6
Enter minutes parked: 34
Amount due ($): $18
```

Example input/output:

```
Enter hours parked: 0
Enter minutes parked: 34
Amount due ($): $0
```

Example input/output:

```
Enter hours parked: 26
Enter minutes parked: 28
Amount due ($): $34
```

Example input/output:

```
Enter hours parked: 50
Enter minutes parked: 2
Amount due ($): $50
```

2. A teacher will assign homework and give the number of days for the students to work on. The student is responsible for calculating the due date. The teacher does not collect homework on Friday or weekend. Write a C program that let the user enter today's day of the week (0 for Sunday, 1 for Monday, etc.) and the number of days to allow the students to do the work, which may be several weeks. Calculate and display the day of the week on which the work would be due. If that day Friday, Saturday, or Sunday– add enough days to the number of days to reach the following Monday. Print the day of the week the work is due and the corrected value of the number of the days.

Example input/output:
Enter today's day of the week: 6
Enter the number of days for doing the work: 15
Output: The due date is Monday. The number of days until due date is 16.

1) Name your program `homework.c`
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 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 the number of days is 11.

Before you submit:

1. Compile with –Wall. –Wall shows the warnings by the compiler. Be sure it compiles on ***student cluster (sc.rc.usf.edu)*** with no errors and no warnings.

   gcc –Wall longterm_parking.c

   gcc –Wall homework.c

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

   chmod 600 longerm_parking.c

   chmod 600 homework.c

3. Test your program with the shell script on Unix:

   chmod +x try_parking

./try_parking

chmod +x try_homework

./try_homework

4. Download the program *longterm_parking.c and homework.c* from student cluster and submit on Canvas>Assignments.

Grading

Total points: 100 (50 points each problem)

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 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`.