Project 6

1. (60 points) Write a program that takes two sets of characters entered by the user and merge them character by character.

```
Enter the first set of characters: dfn h ate
Enter the second set of characters: eedtecsl
Output: defend the castle
```

Your program should include the following function:

```
void merge(char *s3, char *s1, char *s2);
```

The function expects `s3` to point to a string containing a string that combines `s1` and `s2` letter by letter. The first set might be longer or shorter than the second set. The characters in `s1` or `s2` left after merging should be appended to the resulting string `s3`.

1) Assume each input is no longer than 1000 characters.

2) The `merge` function should use pointer arithmetic (instead of array subscripting). In other words, eliminate the loop index variables and all use of the [] operator in the function.

3) To read a line of text, use the `read_line` function (the pointer version) in the lecture notes.

2. (40 points) Write a program that sort its command-line arguments of 10 numbers, which are assumed to be integers. The first command-line argument indicate whether the sorting is in descending order (`-d`) or ascending order (`-a`), if the user enters an invalid option, the program should display an error message. Example runs of the program:

```
./sort -a 5 2 92 424 53 42 8 12 23 41
output: 2 5 8 12 23 41 42 53 92 424

./sort -d 5 2 92 424 53 42 8 12 23 41
output: 424 92 53 42 41 23 12 8 5 2
```

1) Use the selection_sort function provided for project 5. Create another function that's similar but sorts in descending order.
2) Use string library functions to process the first command line argument.
3) Use `atoi` function in `<stdlib.h>` to convert a string to integer form.
4) Compile the program to generate the executable as `sort`:
   `gcc -Wall -o sort command_sort.c`

Before you submit:

1. Compile with –Wall. Be sure it compiles on ***circe*** with no errors and no warnings.

gcc –Wall merge.c
gcc –Wall –o sort command_sort.c

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

chmod 600 merge.c
chmod 600 command_sort.c

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

chmod +x try_merge
./try_merge

chmod +x try_command_sort
./try_command_sort

Total points: 100 (problem 1: 60 points, problem 2: 40 points)

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. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. 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.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **`#define PI 3.141592`**
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: **`tot_vol`** or **`total_volumn`** is clearer than totalvolumn.