Project 3, Program Design

1. Write a program that includes a function `search()` that finds the index of the first element of an input array that contains the value specified. `n` is the size of the array. If no element of the array contains the value, then the function should return `-1`. Name your program `part1.c`. The program takes an int array, the number of elements in the array, and the value that it searches for. The main function takes input, calls the `search()` function, and displays the output.

```
int search(int a[], int n, int value);
```

Example input/output #1:

```
Enter the length of the array: 4
Enter the elements of the array: 8 2 3 9
Enter the value for searching: 3
Output: 2
```

Example input/output #2:

```
Enter the length of the array: 6
Enter the elements of the array: 4 7 1 0 3 9
Enter the value for searching: 5
Output: -1
```

2. Modify the part 1 program so that it deletes all instances of the value from the array. As part of the solution, write and call the function `delete()` with the following prototype. n is the size of the array. The function returns the new size of the array after deletion (The array after deletion will be the same size as before but the actual elements are the elements from index 0 to new_size-1). In writing function `delete()`, you may include calls to function `search()`. The main function takes input, calls the `delete()` function, and displays the output. Name your program `part2.c`.

```
int delete(int a[], int n, int value);
```

Example input/output #1:

```
Enter the length of the array: 6
Enter the elements of the array: 4 3 1 0 3 9
Enter the value for deleting: 3
Output array:
4 1 0 9
```

Before you submit

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

   gcc –Wall part1.c
   gcc –Wall part2.c
2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

   chmod 600 part1.c
   chmod 600 part2.c
3. Test your programs with the shell script on Unix:

   chmod +x try_part1
   ./try_part1

   chmod +x try_part2
   ./try_part2

4. Submit part1.*c* and part2.c on Canvas.

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%
   -Functions implemented as required

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 underscores to make compound names easier to read: **`tot_vol`** or **`total_volumn`** is clearer than totalvolumn.