Project 7

1. (60 points) Write a program to extract Web addresses starting with `www.` and ending with `.edu`. The program displays Web address contained in the input entered by the user. If the input does not contain a web address that starts with `www.` and ends with `.edu`, the program should display a message that indicates such a web address cannot be found.

```
Input: http://www.usf.edu/admission
Output: www.usf.edu

Input: https://www.facebook.com/
Output: Web address starting with www. and ending with .edu not found
```

Your program should include the following function:

```
void extract(char *s1, char *s2);
```

The function expects `s1` to point to a string containing the input as a string and stores the output to the string pointed by `s2`.

1) Name your program `extract.c`.
2) Assume input is no longer than 1000 characters. Assume the input contains no more than one qualifying web address.
3) The `extract` 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.
4) 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 finds either the largest or smallest of the **ten** numbers as command-line arguments. With –l for largest and –s for smallest number, if the user enters an invalid option, the program should display an error message.
Example runs of the program:

```
./find_largest_smallest -l 5 2 92 424 53 42 8 12 23 41
output: The largest number is 424

./find_largest_smallest -s 5 2 92 424 53 42 8 12 23 41
output: The smallest number is 2
```

1) Name your program numbers.c
2) Use `atoi` function in `<stdlib.h>` to convert a string to integer form.
3) Generate the executable as `find_largest_smallest`.

```
gcc -Wall -o find_largest_smallest numbers.c
```

Before you submit:

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

gcc –Wall extract.c
gcc –Wall –o find_largest_smallest numbers.c

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

chmod 600 merge.c
chmod 600 find_largest_smallest.c

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

chmod +x try_extract
./try_extract

chmod +x try_command_line_args
./try_command_line_args

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%
   Problem #1: The `extract` 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. (40%)

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.