CS 350 (Spring 2023)
Assignment 0: Basics of C
Due: 2/1 (11:59pm)

The purpose of this assignment is to reinforce some of the basics of C that we have been learning by writing some code. For this assignment you will write four functions to perform the tasks described below. You should write these functions in a single file, called `hw0.c`. Please write your main function that you test your code with in a separate file. The `hw0.c` file should not contain a main function.

For each function you should not use libraries except for `stdio.h` and `stdlib.h`. In particular, do not use `math.h`. You do not have to consider handling bad inputs for any of the functions, meaning you can expect that the input provided by the user will always be correct. Finally, you can write any helper functions that you would like, as long as the functions that main will call have the signatures given in the problems exactly.

Problem 1: Prime Numbers

Write a function called `prime` with the following header

```
void prime(int n)
```

The function should print out the first `n` prime numbers. The smallest prime number is 2, and you can expect that the smallest value of `n` given will be 1. The numbers in the printout should be separated by a comma and a single space. As an example, if we call

```
prime(1)
```

The printout should be exactly:

```
2
```

As another example, if we call

```
prime(10)
```

The printout should be exactly (note the commas):

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

Problem 2: Remove the vowels

Write a function called `rem` with the following header

```
void rem(const char *s)
```

Remember from lecture that while the argument looks a little odd, `s` is just a character array, and can be accessed by the usual index convention (e.g. `s[0], s[1], etc.`).

This function should print two lines. The first line should just be the given string in `s`. The second line should be the string in `s` without the vowels. For example, if `s` contains the string "Hello World!", then the output should be as follows:

```
Hello World!
Hll Wrld!
```

The next terminal prompt after the printout should not be on the same line as your printout (i.e. remember to include newlines).

Problem 3: Perfect Numbers

Write a function called `perfect` with the following header

```
void perfect(long num)
```

The program should print one of two statements depending on which is true:

- `num` is perfect
- `num` is not perfect

A positive number is perfect if the sum of its factors (excluding itself) is equal to itself. For example, 6 is a perfect number because its factors less than itself are 1, 2 and 3 and $1 + 2 + 3 = 6$. Therefore if the function is called as:

```
perfect(6)
```

The result should be

```
6 is perfect
```

On the other hand, 8 is not perfect, because its factors less than itself are 1, 2, and 4, and $1+2+4 = 7 \neq 8$. Therefore, if the function is called as:

```
perfect(8)
```

The result should be

```
8 is not perfect
```

Problem 4: Nearest Prime

Write a function called nearest with the following header

```
void nearest(void)
```

The function should prompt the user for an input with the following:

```
Enter a positive integer:
```

leaving a space between the colon and the prompt.

The user will enter a positive integer and then press enter, or they will enter EOF by pressing `Ctrl-d`.

If the number entered is prime, the program should respond by printing:

```
The number x is prime.
```

where `x` is the number that the user entered.

Otherwise, if the number is not prime, then there are two options for the nearest prime. Either it will be the next prime number which is larger than the input number, or the previous prime number which is just smaller than the input. Your program should find both of them and then determine which is closer to the input. Then your program should print

```
The nearest prime number to x is y
```

where `x` is the number that the user input, and `y` is the prime number that is closest.

For example, a user might enter

```
Enter a positive integer: 29
```

Then your program should print:

```
The number 29 is prime.
```

As another example, the user might enter:

```
Enter a positive integer: 100
```

Here 100 is not prime. The two prime numbers surrounding it are 97 and 101, and 101 is closer since its distance to 100 is 1, while the distance between 97 and 100 is 3. Therefore, the program should print

```
The nearest prime number to 100 is 101
```

Finally, if a user enters a number that is equidistant from two primes, the smaller number should be returned. For example, the user might enter:

```
Enter a positive integer: 6
```

The two nearest primes are 5 and 7, which are both the same distance from 6 (distance is 1). Therefore, by default the return should be the lower number 5:

```
The nearest prime number to 6 is 5
```

The program should continue to prompt the user in a loop until they give the `EOF`. You should use `getchar()` to get the input from the user. You can safely assume that the integer will have no more than 20 digits.