HOMEWORK-1
Total Points: 40

1. [5 points] Consider the following declaration of a 2-D array.

   ```
   int a[4][3];
   ```

   List all elements (i.e., `a[0][0], a[0][1]`, …, etc.) in the array. If the starting address of the array is 1000, compute the address of each element in the array. Which address will be written when you try to execute the statement "`a[1][7] = 100`"?

Element	Address
a[0][0]	1000
a[0][1]	1004
a[0][2]	1008
a[1][0]	1012
a[1][1]	1016
a[1][2]	1020
a[2][0]	1024
a[2][1]	1028
a[2][2]	1032
a[3][0]	1036
a[3][1]	1040
a[3][2]	1044

   Correct addresses for all elements give 3 points; otherwise, 0

   The address written for a[1][7] is 1040 => give 2 points for the correct answer

2. [5 points] Write a definition of the `strcmp` routine that takes two character arrays containing '\0' terminated strings. It returns 0 if the character strings are equal; otherwise, it returns 1. The prototype of the `strcmp` routine is as follows:
`int strcmp(char str1[], char str2[])`

```
int strcmp(char str1[], char str2[]) {
   int i = 0;
   while (str1[i] == str2[i] && str1[i] != '\0') {
     i++;
   }
   if (str1[i] == str2[i])
      return 0;
   else
      return 1;
}
```

Give up to 2 points if the solution is partially correct.

3. [20 points] Consider the definition of the power function below, as discussed in class.

```
int power(int x, int n) {
  int pow_h;
  if (n == 0)                  // one comparison operation
    return 1;
  if ((n % 2) == 0) {          // one comparison operation
    pow_h = power(x, n/2);
    return pow_h * pow_h;      // one multiplication operation
  }
  else {
    pow_h = power(x, (n-1)/2);
    return x * pow_h * pow_h; // two multiplication operations
  }
}
```

How many comparisons, multiplications, and function calls will occur during the computation of `power(3, 23)`?

Now rewrite the definition of power that roughly computes `x^(n/3)` instead of `x^(n/2)` during the recursive step, and add more conditions to compute the correct value of `x^n`, where `^` is the power operator.

How many comparisons, multiplications, and function calls will occur during the computation of `power(3, 23)` in your modified implementation?

Number of comparisons for power(3, 23): 11 => 1 point for the correct answer
Number of multiplications for power(3, 23): 9 => 1 point for the correct answer
Number of function calls for power(3, 23): 6 (5 may also be acceptable if they are not counting the initial call to power(3, 23)) => 1 point for a correct answer

Implementation of new power function:

```
int power(int x, int n) {
   int pow_h;
   if (n == 0)
      return 1;
   if ((n % 3) == 0) {
      pow_h = power(x, n/3);
      return pow_h * pow_h * pow_h;
   }
   else if ((n%3) == 1) {
      pow_h = power(x, (n-1)/3);
      return x * pow_h * pow_h * pow_h;
   }
   else {
      pow_h = power(x, (n-2)/3);
      return x * x * pow_h * pow_h * pow_h;
   }
}
```

Correct implementation of the power function: 14 points
Give up to 5 points if the solution is partially correct.

The below answers may change if the above implementation is different. Kindly correlate with the actual answer. If the implementation is incorrect, don't give any points for this part.
Number of comparisons in new implementation: 10 => 1 point for the correct answer
Number of multiplications in new implementation: 11 => 1 point for the correct answer
Number of function calls in new implementation: 4 (or 3 if the initial call is not counted) => 1 point for a correct answer

4. [10 points] Consider the definition of the `fib` function below, as discussed in class.

```
int fib(int n) {
  if (n == 0 || n == 1)
    return n;
  return fib(n-1) + fib(n-2);
```

```
}
```

As discussed, during the execution of the `fib(4)`, the following function calls will be made in the given order.

```
fib(4), fib(3), fib(2), fib(1), fib(0), fib(1), fib(2), fib(1),
fib(0)
```

Write the sequence of function calls that will be made during the execution of `fib(6)` in the correct order.

```
fib(6), fib(5), fib(4), fib(3), fib(2), fib(1), fib(0), fib(1),
fib(2), fib(1), fib(0), fib(3), fib(2), fib(1), fib(0), fib(1),
fib(4), fib(3), fib(2), fib(1), fib(0), fib(1), fib(2), fib(1),
fib(0)
```

Zero marks if the order is incorrect or a function call is missing. The first fib(6) can be omitted.