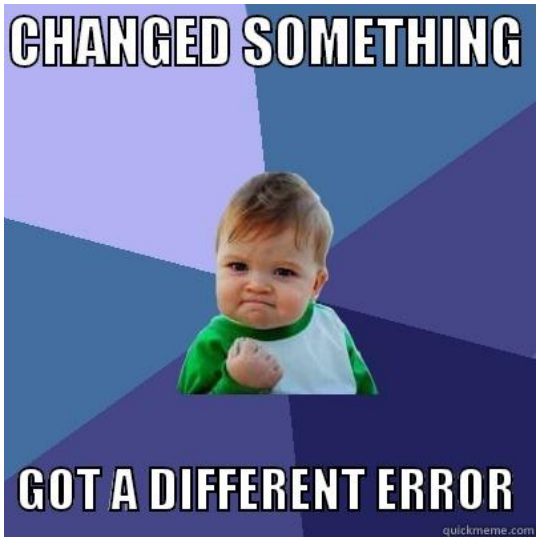
Problem Set #4

CSEE 3827: Fundamentals of Computer Systems



- For each function, place all of your code above the line in the scaffolding that begins

```
#### Do not remove this separator. ...
```

- We will test your code using the main function in the scaffolding.
- We will also test that your code adheres to calling conventions by calling it from other versions of main that have different register usage (but adhere to conventions). If you follow the information about register usage provided in the prompts, your code will adhere to conventions.
- To submit, you should upload three files named ifib.s, minbills.s, and fizzbuzz.s to gradescope.

1. **ifib** Write a function in MIPS that takes one integer argument and returns the Fibonacci number corresponding to that argument. Your code should handle non-negative integer argument values, including zero. No need to error check for negative arguments. The iterative Fibonacci algorithm in C is:

```
int ifib(int a) {
    int x = 0;
    int y = 1;
    int z = 0;
    for (int i = 0; i < a; i++) {
       z = x + y;
       x = y;
       y = z;
    }
    return x;
}
```

```
ifib:
    # At the start of fibonacci, the argument can be found in $a0.

    # Replace the line below with your code
    li $v0, 0

return:
    # Your return value should be in $v0 prior to returning
    jr $ra
```

2. **minbills()** Implement the following function in MIPS.

```
int minbills(int totalvalue) {
    int count = 0;
    int currvalue = totalvalue;
    while (currvalue >= 10) {
        count += 1;
        currvalue -= 10;
    }
    while (currvalue >= 5) {
        count += 1;
        currvalue -= 5;
    }
    while (currvalue >= 1) {
        count += 1;
        currvalue -= 1;
    }
    return count;
}
```

To ensure your code adheres to calling conventions, use only temporary registers and the ones specified in the scaffolding.

```
minbills:
    # At the start of minbills the argument can be found in $a0.

    # Replace the line below with your code
    li $v0, 0

return:
    # Your return value should be in $v0 prior to returning.
    jr $ra
```

3. **fizzbuzz()** Implement a function that prints out every number from 1 to the input argument. However, you must print the word 'fizz' instead of numbers that are multiples of 3 and the word 'buzz' instead of numbers that are mutiples of 5. If a number is a multiple of both 3 and 5, print 'fizzbuzz'. The printing functions are provided, along with some additional scaffolding code to allow your subcalls to the print functions to work. As before, use only `$t` registers unless otherwise specified, in order to adhere to calling conventions.

   Note that the scaffolding provides specifics of how to call the printer functions:

```
fizzbuzz:
    # Do not remove this code.  It is necessary for the subcalls to the print functions.
    addi $sp, $sp, -4
    sw   $ra, 0($sp)

    # Put your code here.
    # To print an integer, put the integer in $a0 call the helper:
    #    jal print_int
    # To print a string, make the appropriate function call:
    #    jal print_fizz
    #    jal print_buzz
    #    jal print_fizzbuzz

return_from_fizzbuzz:
    # return to calling function
    lw   $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra
```