# CS136 EXERCISE SOLUTIONS: MODULE 1

## Running C in edX; introducing printf

```c
#include <stdio.h>

int main(void) {
  printf("Hello world!\n");
  printf("@\n");
  printf("|\n");
  printf("v\n");
  printf("|\n");
}
```

```c
#include <stdio.h>

int main(void) {
  printf("%d\n", 4 * (7 + 3) / (1 + 1));
}
```

## Working with digits

```c
#include "cs136.h"

int last_digit(int n) {
  return n % 10;
}

int rest_digits(int n) {
  return n / 10;
}

int main(void) {
  assert(last_digit(136) == 6);
  assert(rest_digits(136) == 13);
}
```

## IF STATEMENTS

```c
#include "cs136.h"

int sum_to(int n) {
  trace_int(n);
  if (0 == n) {
    return 0;
  } else {
    return n + sum_to(n - 1);
  }
}

int main(void) {
  assert(sum_to(5) == 15);
  printf("All tests passed!\n");
}
```

## Practice writing recursive C

```c
// sum_divisible(n, d0, d1) Return the sum of natural numbers to n
//     that are divisible by d0 or d1.
int sum_divisible(int n, int d0, int d1) {
  if (n == 0) {
    return 0;
  } else if (0 == n % d0 || 0 == n % d1) {
    return n + sum_divisible(n - 1, d0, d1);
  } else {
    return  sum_divisible(n - 1, d0, d1);
  }
}
```

```c
// has_divisor_below(n, d) Determine if any number between 2 and d divides n.
// Requires: n >= 0, d >= 0
bool has_divisor_below(int n, int d) {
  if (d < 2) {
    return false;
  } else if (0 == n % d) {
    return true;
  } else {
    return has_divisor_below(n, d - 1);
  }
}

// is_prime(n) Determine if n is prime.
// Requires: n >= 0
bool is_prime(int n) {
  return n >= 2 && !has_divisor_below(n, n - 1);
}
```

## Using trace_int

```
// fib(n) Calculate the n-th Fibonacci number.
// Requires: n >= 0
int fib(int n) {
  assert(n >= 0);

  trace_int(n);

  if (n < 2) {
    return n;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

## Accumulative Recursion

```
int sum_to_acc(int n, int acc) {
  if (0 == n) {
    return acc;
  } else {
    return sum_to_acc(n - 1, acc + n);
  }
}

int sum_to(int n) {
  return sum_to_acc(n, 0);
}
```

```
// fact_acc(n, acc) Return acc * n!
// Requires: n >=0
int fact_acc(int n, int acc) {
  assert(n >= 0);
  if (0 == n) {
    return acc;
  } else {
    return fact_acc(n-1, n * acc);
  }
}

// fact(n) Calculate n!.
// Requires: n >= 0
int fact(int n) {
  assert(n >= 0);
  return fact_acc(n, 1);
}
```

```
int fib_acc(int n, int f0, int f1) {
  assert(n >= 0);
  if (n == 0) {
    return f0;
  } else {
    return fib_acc(n - 1, f1, f0 + f1);
  }
}

int fib(int n) {
  assert(n >= 0);
  return fib_acc(n, 0, 1);
}
```