PE Cheatsheet

Debug Spaces

```
./onigiri | sed 's/ /./g'
```

Typecasting

```
cs1010_print_double((double)mcneal*100.0/(double)total);
```

Convert Double to Long

```
long numerator = (long)(round(num * 10));
```

Ceiling Function For Long

```
double num = ceil((double)distance / (double)1000);
long segments = (long)(round(num));
```

Comparing Double Variable to Number

```
while (fabs(fx) >= 0.000000001) {
    double fpx = fp(a, b, c, x);
    x = fx/fpx;
    fx = f(a, b, c, d, x);
}
```

Is_prime Algorithm

```
bool is_prime(long n)
{
    if (n < 2){
        return false;
    }
    for (int i = 2; i * i <= n; i += 1) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
sad
```

Leap Year

- years that are multiples of 4 except for years divisible by 100 but not by 400

```
bool is_leap_year(long year){
  if (year % 100 == 0 && year % 400){
    return false;
  }
  return year % 4 == 0;
}
```

Compares Date

```
bool is_greater(long month_1, long day_1, long month_2, long day_2){
  if (month_1 > month_2){
    return true;
  }
  if (month_1 == month_2){
    return day_1 > day_2;
  }
  return false;
}
```

GCD

```
long gcd(long a, long b) {
  if (b == 0) {
    return a;
  }
  return gcd(b, a % b);
}
```

Prime Factorization

```
long curr = n;
if (is_prime(curr)) {
  print(curr, 1);
    return;
}
for (long factor = 2; factor <= sqrt(curr) && curr != 1; factor += 1) { // <changed here
  if (is_prime(factor)) {
    long count = 0;
      while (curr % factor == 0) {
        curr = curr / factor;
          count += 1;
      }
    if (count != 0) {
      print(factor, count);
    }
    if (is_prime(curr) && curr != 1) {
      print(curr, 1);
```

```
        return;
    }
  }
}
```

X power y Recursively

```
long compute_power(long x, long y){
  if (y == 0){
    return 1;
  }
  if (x == 0){
    return 0;
  }
  return compute_power(x, y - 1) * x;
}
```

Binary to Decimal

```
long binary_to_decimal(long binary){
  long decimal = 0;
  long power = 0;
  while (binary > 0){
    long last_digit = binary % 10;
    decimal += (unsigned long)last_digit << (unsigned long)power;
    power += 1;
    binary /= 10;
  }
  return decimal;
}
```

Onigiri

```
ooiwt@pe112:~/ex02-ooiwt$ ./onigiri
5
    #
   ###
  #####
 #######
#########
ooiwt@pe112:~/ex02-ooiwt$ ./onigiri
3
  #
 ###
#####
ooiwt@pe112:~/ex02-ooiwt$ ./onigiri
1
#
```

```
void draw_onigiri(long h){
  for (long i = 0; i < h; i += 1){
    for (long j = 0; j < h * 2 - 1; j += 1){
      long mid = (h * 2 - 1) / 2;
      if (j < (mid - i) || j > (mid + i)){
        cs1010_print_string(" ");
      } else {
        cs1010_print_string("#");
      }
    }
    cs1010_println_string("");
  }
}
```

Fibnoacci (Iterative)

```
long fib(long n){
  if (n == 1 || n == 2){
    return 1;
  }
  long current_num = 0;
  long previous_num = 1;
  long previous_two_num = 1;
  for (long i = 3; i <= n; i += 1){
    current_num = previous_num + previous_two_num;
```

```
    previous_two_num = previous_num;
    previous_num = current_num;
  }
  return current_num;
}
```

Nine

- Looks for the least significance occurrence of digit 9 in a given digit.
- Rightmost digit has the position of 1
- If doesn't appear –> return 0

```
long find_least_significant_9(long n){
  long position = 1;
  while (n > 0){
    if (n % 10 == 9){
      return position;
    }
    position += 1;
    n /= 10;
  }
  return 0;
}
```

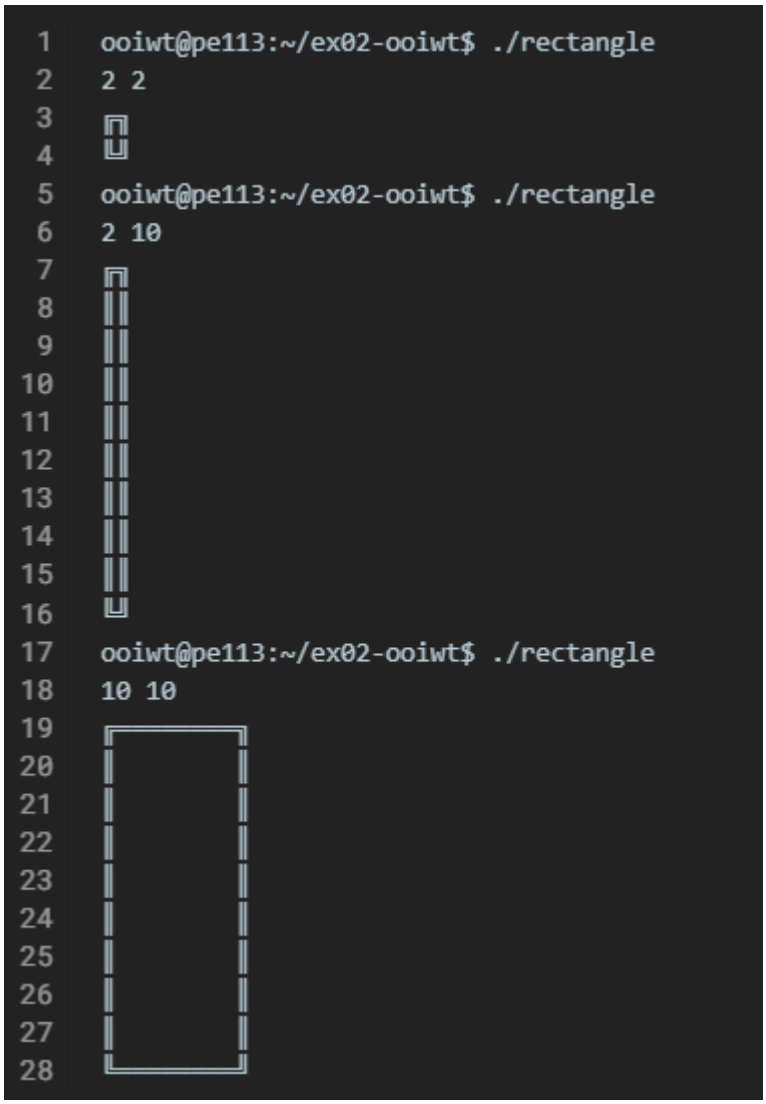
Draw HDB (Iterative)

```
ooiwt@pe119:~/2122s1/ex02-ooiwt$ ./hdb
3 3
###
###
###
ooiwt@pe119:~/2122s1/ex02-ooiwt$ ./hdb
10 15
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
ooiwt@pe119:~/2122s1/ex02-ooiwt$ ./hdb
24 10
########################
########################
########################
########################
########################
########################
########################
########################
########################
########################
```

```
void draw_hdb(long w, long h){
  for (long i = 0; i < h; i += 1){
```

```
    for (long j = 0; j < w; j += 1){
      cs1010_print_string("#");
    }
    cs1010_println_string("");
  }
}
```

Rectangle




```
void draw_rectangle(long width, long height){
  for (long i = 0; i < height; i += 1){
```

```
    for (long j = 0; j < width; j += 1){
      if (i == 0 && j == 0){
        cs1010_print_string(TOP_LEFT);
      } else if (i == 0 && j == width - 1){
        cs1010_print_string(TOP_RIGHT);
      } else if (i == (height - 1) && j == 0){
        cs1010_print_string(BOTTOM_LEFT);
      } else if (i == (height - 1) && j == (width - 1)){
        cs1010_print_string(BOTTOM_RIGHT);
      } else if (i == 0 || i == (height - 1)){
        cs1010_print_string(HORIZONTAL);
      } else if (j == 0 || j == (width - 1)){
        cs1010_print_string(VERTICAL);
      } else {
        cs1010_print_string(" ");
      }
    }
    cs1010_println_string("");
  }
}
```

Recursion

Sum of the cube of digits in Integer Recursively

```
long sum_of_digits_cubed(long num){
  long digit = num % 10;
  long digit_cubed = digit * digit * digit;
  if (num <= 9){
    return digit_cubed;
  }
  return sum_of_digits_cubed(num / 10) + digit_cubed;
}
```

Rearranging Digits to have Largest

```
long insert(long n, long digit)
{
  if (n % 10 > digit) {
    return n * 10 + digit;
  }
  if (n == 0) {
    return digit;
  }
  return (insert(n / 10, digit) * 10) + (n % 10);
}
long largest(long n)
```

```
{
  if (n < 10) {
    return n;
  }
  return insert(largest(n / 10), n % 10);
}
```

- For example, the largest possible number we get by rearranging the digits in 6752378 is 8776532. The largest possible number we get by rearranging the digits in -1010 is -11
- For negative numbers, we need to rearrange the digits in reverse order.

Find Longest Consecutive Digit from Digit Sequence

```
long longest_count = 1 ;
long longest_digit;
long current_count = 0;
long current_digit = n % 10;

do {
// Increase the counter if we see the same digit.
// Otherwise reset counter to 1.
    if (n % 10 == current_digit) {
        current_count += 1;
    } else {
        current_count = 1;
    }
    // Checks if we find a longer (or equally long)
    // consecutive sequence. Update longest_digit
    // and longest_count if so.
    if (current_count > longest_count) {
        longest_digit = current_digit;
        longest_count = current_count;
    } else if (current_count == longest_count) {
        if (current_digit < longest_digit) {
            longest_digit = current_digit;
        }
    }

    // Update the current digit to the last digit of n
    // and shorten n by one digit.
    current_digit = n % 10;
    n = n / 10;
} while (n > 0);
return longest_digit;
}
```

Insert Digit Based on Index Recursively

```
long insert(long N, long d, long k){
  if (k == 1){
    return N * 10 + d;
  }
  return insert(N / 10, d, k - 1) * 10 + (N % 10);
}

#In Main to deal with negative numbers
  if (N >= 0){
   answer = insert(N, d, k);
  }
  else {
    answer = -insert(-N, d, k);
  }
```

- Base case
  - When the index to insert is 1 –> the number is * 10 and added with the insertion number
- Solving sub-problem
  - insert the shorter number (number / 10) with index (k-1)

Reverse Digit Recursively

```
long num_of_digits(long n){
  if (n < 10){
    return 1;
  }
  return 1 + num_of_digits(n / 10);
}

long pow10(long k){
  if (k == 0){
    return 1;
  }
  return 10 * pow10(k - 1);
}

long reverse(long n, long digits){
  if (digits == 1){
    return n;
  }
  return (n % 10) * pow10(digits - 1) + reverse(n / 10, digits - 1);
}

long output = reverse(n, num_of_digits(n));
```

Simple Digits

```
long simplify(long n) {
  if (n / 10 == 0) {
    // one digit remaining
    return n;
  }
  if ((n / 10) % 10 == n % 10) {
    // last two digits are the same
    return simplify(n / 10);
  }
  // last two digits are different
  return simplify(n / 10) * 10 + (n % 10);
}
```

Useful Tricks

Print Characters k times

```
void print_k_times(long times, char c) {
    for (long i = 0; i < times; i++) {
        putchar(c);
    }
}
```

- Useful for drawing patterns

Aligning Fraction

```
ooiwt@pe101:~$ ./fraction
3 5 3 5
 1
1-
 5
ooiwt@pe101:~$ ./fraction
1 111 1 111
 2
---
111
ooiwt@pe101:~$ ./fraction
50010 10000 30017 10000
  27
8-----
 10000
```

```
long ilen;
if (whole == 0) {
  ilen = 0;
} else {
  ilen = count_digits(whole);
}
long nlen = count_digits(numerator);
long dlen = count_digits(denominator);
print_k_times(ilen + (dlen nlen)/2, ' ');
cs1010_println_long(numerator);
if (whole != 0) {

  cs1010_print_long(whole);
}
print_k_times(dlen, ' ');
cs1010_println_string("");
print_k_times(ilen, ' ');
cs1010_println_long(denominator);
```

Taxi

```
// Return true if first time input is larger or equals the second.
bool is_greater_equals(long hour_1, long min_1, long hour_2, long min_2){
  if (hour_1 > hour_2){
    return true;
  }
  if (hour_1 == hour_2){
    return min_1 >= min_2;
  }
  return false;
}

// The number of segments required.
 */
long find_segments(long distance, long segment_size){
  if (distance % segment_size){
    return distance / segment_size + 1;
  }
  return distance / segment_size;
}

double determine_surcharge(long day, long hour, long min){
  if ((day >= 1 && day <= 5 && is_greater_equals(hour, min, 6, 0)
        && is_greater_equals(9, 29, hour, min))
      || (is_greater_equals(hour, min, 18, 0) && is_greater_equals(23, 59, hour, min))){
    return 1.25;
  }
  if (is_greater_equals(hour, min, 0, 0) && is_greater_equals(5, 59, hour, min)){
    return 1.5;
  }
  return 1;
}

double fare_calc(long distance){
  double fare = 0;
  if (distance <= 1000){
    fare = 3.9;
  } else if (distance >= 1001 && distance <= 10000){
    long num_of_segments = find_segments((distance - 1000), 400);
    fare = 3.9 + (double)num_of_segments * 0.24;
  } else {
    long num_of_segments = find_segments((distance - 10000), 350);
    fare = 3.9 + 23 * 0.24 + (double)num_of_segments * 0.24;
  }
  return fare;
```

```
}

int main()
{
  long day = cs1010_read_long();
  long hour = cs1010_read_long();
  long min = cs1010_read_long();
  long distance = cs1010_read_long();
  double fare = fare_calc(distance);
  double surcharge_factor = determine_surcharge(day, hour, min);
  double final_fare = fare * surcharge_factor;
  cs1010_println_double(final_fare);
}
```

Draw Square

```
ooiwt@pe101:~$ ./square
4
####
#  #
#  #
####
```

Sample Run 5

```
ooiwt@pe101:~$ ./square
5
#####
#   #
# # #
#   #
#####
```

Sample Run 6

```
ooiwt@pe101:~$ ./square
6
######
#    #
# ## #
# ## #
#    #
######
```

Sample Run 7

```
ooiwt@pe101:~$ ./square
7
#######
#     #
# ### #
# # # #
# ### #
#     #
#######
```

Sample Run 8

```
ooiwt@pe101:~$ ./square
10
##########
#        #
# ###### #
# #    # #
# # ## # #
# # ## # #
# #    # #
# ###### #
#        #
##########
```

```
void print_line(long width) {
    for (long i = 0; i < width; i+=1) {
        cs1010_print_string("#");
    }
}

void print_border(long width) {
    cs1010_print_string("#");
    for (long i = 0; i < width 2; i+=1) {
        cs1010_print_string(" ");
    }
    cs1010_print_string("#");
}

void print_square(long row, long width) {
    if (width == 1) {
        cs1010_print_string("#");
    } else if (row == 0 || row == width 1) {
        print_line(width);
    } else if (row == 1 || row == width 2) {
        print_border(width);
    } else {
        cs1010_print_string("# ");
        print_square(row 2, width 4);
        cs1010_print_string(" #");
    }
}

int main()
{
    long width = cs1010_read_long();
    for (long row = 0; row < width; row += 1) {
    print_square(row, width);
    cs1010_println_string("");
    }
}
```

Pattern

The inputs given are an interval m ($m \geq 1$) and the height of the triangle h.

The triangle has h rows. The first row of the triangle has one cell, the second row has three cells, the third row has five, etc. The cells are centrally aligned so that visually they form an equilateral triangle. We call the left-most cell of each row the leading cell.

Each cell in the triangle contains m integers. The first cell in the first row contains the numbers 1, 2, ..., m. The leading cell of the next row, Row 2, contains m numbers between $m+1$ and $3m$, with an increment of 2: i.e., $m+1$, $m+3$, $m+5$, .., $m+(2m-1)$. The leading cell of the next row, Row 3, contains the numbers $3m+1$ and $6m$, with an increment of 3: i.e., $3m+1$, $3m+4$, $3m+7$,.. $3m+(3m-2)$, etc.

```
long find_leading_num(long m, long row) {
</strong>  if (row == 1){
    return 1;
  }
  return find_leading_num(m, row - 1) + ((row - 1) * m);
}

//Prints the padding of a given row.
void print_spaces(long h, long row){
  long width = (h * 2) - 1;
  long num_of_cells = (row * 2) - 1;
  long num_of_spaces = (width - num_of_cells) / 2;
  for (long i = 1; i &#x3C;= num_of_spaces; i += 1) {
    cs1010_print_string(" ");
  }
}

//Prints the parallax compression row.
void draw_row(long m, long h, long row){
  print_spaces(h, row);
  long leading_num = find_leading_num(m, row);
  long num = leading_num;
  for (long i = 1; i &#x3C;= 2 * row - 1; i += 1) {
    bool has_prime = false;
    for (long j = 1; j &#x3C;= m; j += 1) {
      if (is_prime(num)) {
        has_prime = true;
      }
      num += row;
    }
    if (has_prime) {
      cs1010_print_string("#");
```

```
    } else {
      cs1010_print_string(".");
    }
    num = leading_num + i;
  }
  print_spaces(h, row);

}

// Prints the parallax compression pattern.
void draw_parallax_compression(long m, long h) {
  for (long row = 1; row &#x3C;= h; row += 1) {
    draw_row(m, h, row);
    cs1010_println_string("");
  }
}
```