**Submit the following entries in a word file:**

**Problem Statement:**

**Problem Statement:** How to simulate an n-sided coin using a 2 sided coin. (Solve for n=6).

**Algorithm:**

**Solution:** You can simulate an n-sided coin using a two sided coin as follows:

Let m = . The base is always 2. (Example, for n = 6, m = 3)

Flip a 2-sided coin m times and record the result of every flip. (HHT may be represented as 110)

Convert the binary number generated to a decimal number. (Example: (110)2 = (6)10 )

Repeat for the number of sample points required.

**Challenge:**

If m =3, the numbers generated will be in the range 0 to 7, whereas we need the numbers in the range (1, 6).

**A possible Solution-**

* When you get a number not in range, ignore it and regenerate another number in range.   
  In this example – When you generate a 0 or a 7, ignore it and generate another number till you get a number in the range and record that.

**Note:** When n = 6, we can simulate a dice using a 2-sided coin.

**C Code**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

int flip\_coin(){

return rand() % 2;

}

int flip\_and\_convert(int i){

return (int)(pow(2, i) \* flip\_coin());

}

int simulate\_dice(int n){

int result;

do{

result = 0;

int m = (int)ceil(log2(n));

for(int i = 0; i < m; i++)

result += flip\_and\_convert(i);

} while(result == 0 || result > n);

return result;

}

int main(){

int seed = time(NULL);

srand(seed);

// Roll a 6 Sided Die simulated by a Coin

int sample[7] = {0};

for(int i = 0; i < 1000; i++)

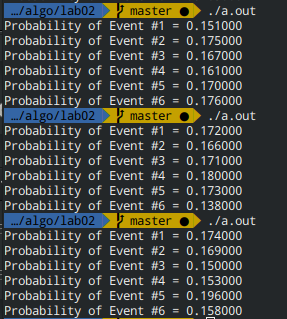
sample[simulate\_dice(6) - 1] += 1;

for(int i = 0; i < 6; i++)

printf("Probability of Event #%d = %f\n", i+1, sample[i]/1000.0);

return 0;

}



**Result Table**

Sample space is the following: {1, 2, 3, 4, 5, 6}

Find the probability of each event, while generating 1000 samples points.

|  |  |
| --- | --- |
| Event | Probability of event |
| 1 | 0.172 |
| 2 | 0.166 |
| 3 | 0.171 |
| 4 | 0.180 |
| 5 | 0.173 |
| 6 | 0.158 |

**Analysis**

**Did the result meet the expectation?**

**If no, can you think of an improvement?**

The expectation was 0.1667

The results are within acceptable error ranges.

Although, due to the random nature of the generation, sometimes there is a big difference in probability for small n as seen in the screenshot above. By increasing n we can reach closer to the actual values.