Arunima Singh Thakur, 180905218, C, 31, CSEPP Lab End Sem, [illegible], 11th June 2021

```
~~// host code~~
~~int main() {~~
~~    // All the declarations~~
~~    int n, i, j;~~
~~    char **mat, *inp;~~
~~    // Scanning the input~~
~~    scanf("%d", &n);~~
~~    char res[n][n];~~
~~    // Initializing the matrix & reading input~~
~~    mat = (char**) malloc(n * sizeof(char*));~~
~~    for (i=0; i<n; i++)~~
~~    {~~
~~        scanf("%s", inp);~~
~~        mat[i] = (char) malloc(strlen(inp) *~~
~~                         sizeof(char*));~~
~~        strcpy(mat[i], inp);~~
~~    }~~
~~    int size = n * n * sizeof(char);~~

int main()
{
    int N;
    printf(" Enter an even no: ");
    scanf("%d", &N);
```

```
if(N%2!=0) return -1;
char A[n][n], str[N], Res[N][N];
printf("Enter %d words =\n", N);
for(int i=0 ; i<N; i++)
{
    printf("Enter word %d : ", (i+1));
    scanf("%s", str);
    if(strlen(str) <N)
        strcpy(A[i], str);
    else
    {
      printf("Enter again! Less than
                        %d reqd.", N);
      i--;
    }
}
char *d_A, *d_Res;
int size = sizeof(char) *N *N;
cudaMalloc((void **) &d_A, size);
cudaMalloc((void **) &d_Res, size);
~~cudamemcpy(d_A, A, size, cudamemcpyHost~~
            ~~cudaMemoryToDevice);~~
cudamemcpy(d_A, A, size,
        cudamemoryHostToDevice);
dim3 gridDim(2,2);
dim3 blockDim(ceil(N/2.0), ceil(N/2.0));
answer<<<gridDim, blockDim>>>(d_A,
                        d_Res, N);
~~cuda~~ cudaMemcpy(Res, d_Res, size,
        cudamemcpyDeviceToHost);
```

```
printf("Resultant array:\n");
for(int i=0; i<N; i++)
{
    for(int j=0; j<N; j++)
        printf("%c", Res[i][j]);
    printf("\n");
}
cudaFree(d_A);
cudaFree(d_Res);
return 0;
}
```

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <cuda.h>
#include "cuda_runtime.h"
#include "device_launch_parameter.h"
#include <string.h>
__global__ void answer(char *A, char *Res,
            int N)
{
    int blockId = blockIdx.x + blockIdx.y *
                                  gridDim.x;

    int threadId = blockId * (blockDim.x *
        blockDim.y) + (threadIdx.y * blockDim.x)
            + threadIdx.x;

    int row = threadIdx.y + blockIdx.y *
                              blockDim.y;

    int col = threadIdx.x + blockIdx.x *
                              blockDim.x;
```

```
if(row == 0 || col == 0 || row == N-1 ||
            col == N-1)
      d_Res[row*N+col] = '!';
  else if(d_A[row * N + col != '\0'){
{ ~~else~~ if((d_A[row * N + col] >= 'a' &&
          dA[row * N + col] <= 'z') ||
    (d_A [row * N +col] >= 'A' &&
              dA[row * N + col] <= 'Z'))
{
      int idx = row * N + col;
      if(dA[idx] == 'a' || d_A [idx] == 'e'||
       d_A[idx] == 'i' || d_A [idx] == 'o' ||
      d_A [idx] == 'u')
            d_Res [idx] = d_A[idx]-32;
     else if(d_A [idx] == 'A' || d_A [idx] == 'E' ||
      dA[idx] == 'I'|| d_A[idx] == 'O' || d_A[idx]
         == 'U')
            d_Res[idx] = d_A[idx] +32;
     else
            d_Res [idx] = d_A [idx];
   }
}
 else
{
     int c = 0;
     for (int i = 2; i < row; i++)
           if(row % i == 0) c++;
     if(c == 0)
           d_Res [row * N + col] = '*';
     else
```

d_Res[row * N + col] = '#';

}

~~for(int i=0; i<row; i++)~~

}
~~{~~

~~~~scanf("~~ printf("\n");~~

}

~~for(int j=0; j<col; j++)~~

~~printf("%d", d_Res[~~row * N + col~~~~
~~~~