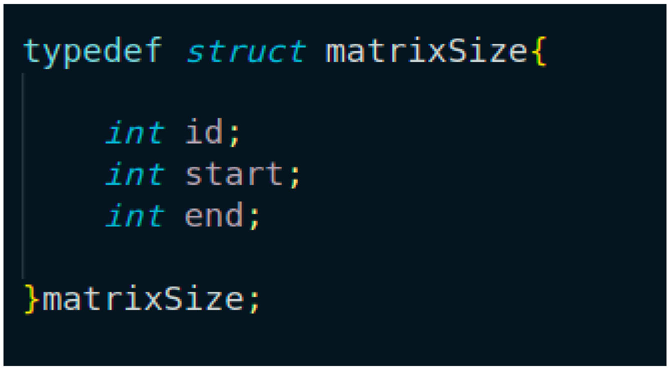
CSE 344 SYSTEM PROGRAMMING REPORT:

-SERDİL ANIL ÜNLÜ

-1801042672

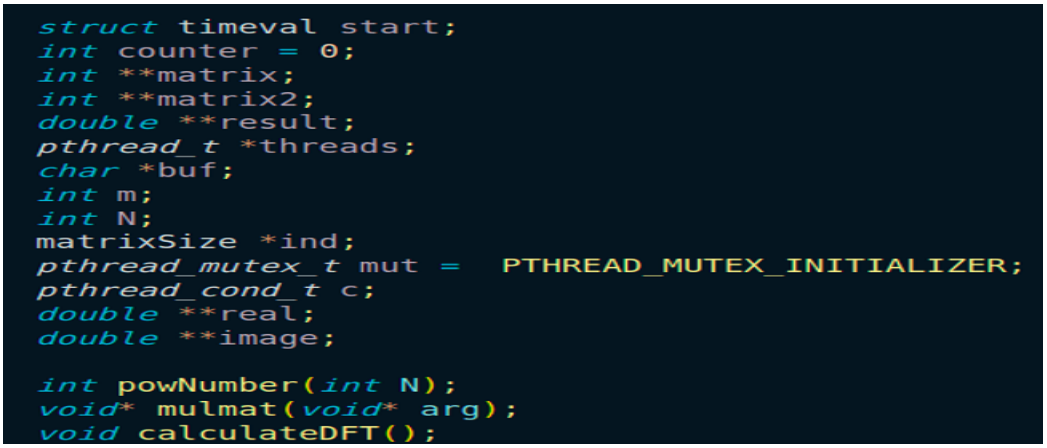
HOW I SOLVED THIS PROBLEM?

- I created a structure to do the starting, ending and naming of threads. I used the start and end values to share the matrix size of the threads. I used the id value to print which thread is which thread.



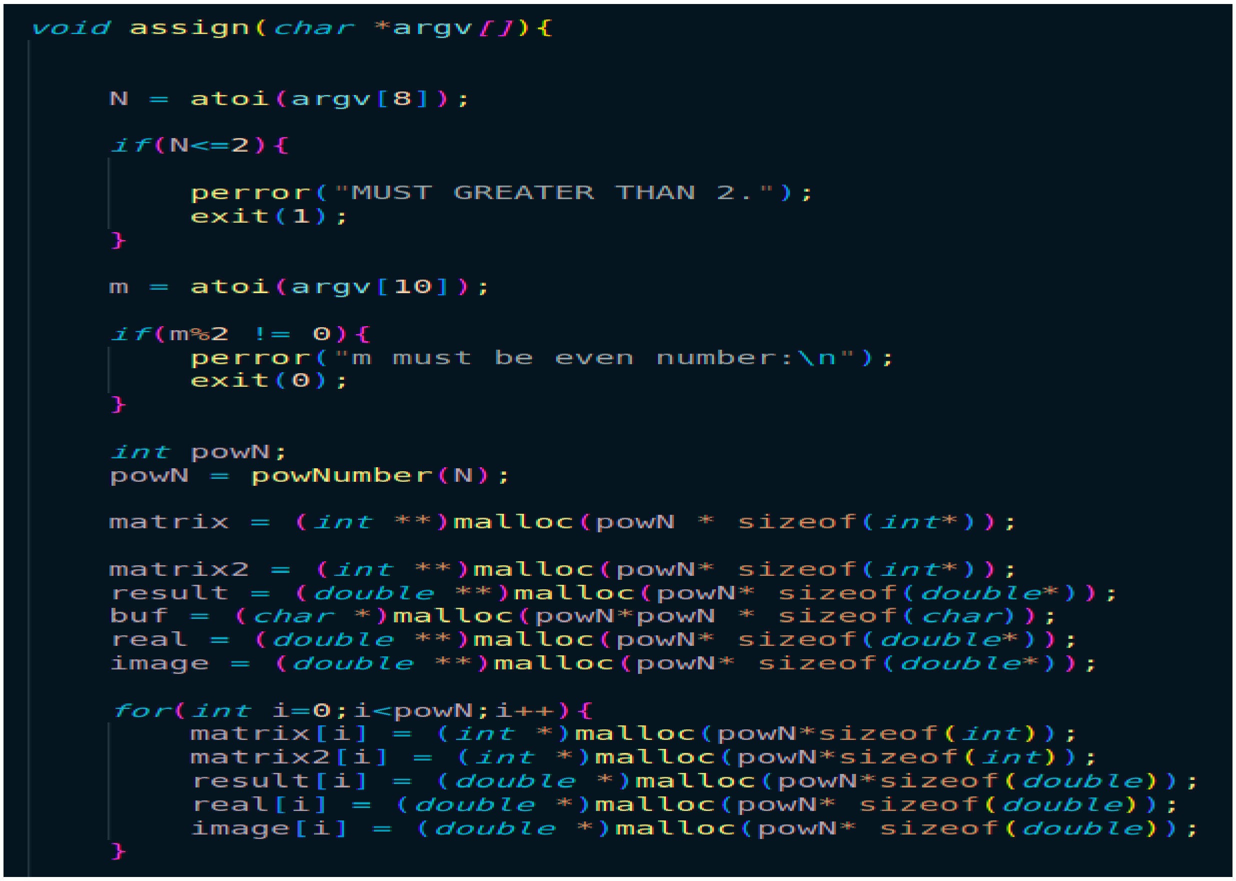
```c
typedef struct matrixSize{

    int id;
    int start;
    int end;

}matrixSize;
```

- I kept the matrix size, number of threads, threads, multiplication and fourier results of the matrices as global values in order to reach them more easily.



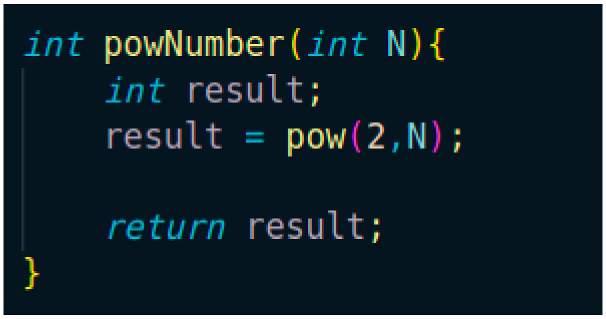
```c
struct timeval start;
int counter = 0;
int **matrix;
int **matrix2;
double **result;
pthread_t *threads;
char *buf;
int m;
int N;
matrixSize *ind;
pthread_mutex_t mut =  PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c;
double **real;
double **image;

int powNumber(int N);
void* mulmat(void* arg);
void calculateDFT();
```

- assign() function:



```
void assign(char *argv[]){

    N = atoi(argv[8]);

    if(N<=2){

        perror("MUST GREATER THAN 2.");
        exit(1);
    }

    m = atoi(argv[10]);

    if(m%2 != 0){
        perror("m must be even number:\n");
        exit(0);
    }

    int powN;
    powN = powNumber(N);

    matrix = (int **)malloc(powN * sizeof(int*));

    matrix2 = (int **)malloc(powN* sizeof(int*));
    result = (double **)malloc(powN* sizeof(double*));
    buf = (char *)malloc(powN*powN * sizeof(char));
    real = (double **)malloc(powN* sizeof(double*));
    image = (double **)malloc(powN* sizeof(double*));

    for(int i=0;i<powN;i++){
        matrix[i] = (int *)malloc(powN*sizeof(int));
        matrix2[i] = (int *)malloc(powN*sizeof(int));
        result[i] = (double *)malloc(powN*sizeof(double));
        real[i] = (double *)malloc(powN* sizeof(double));
        image[i] = (double *)malloc(powN* sizeof(double));
    }
```

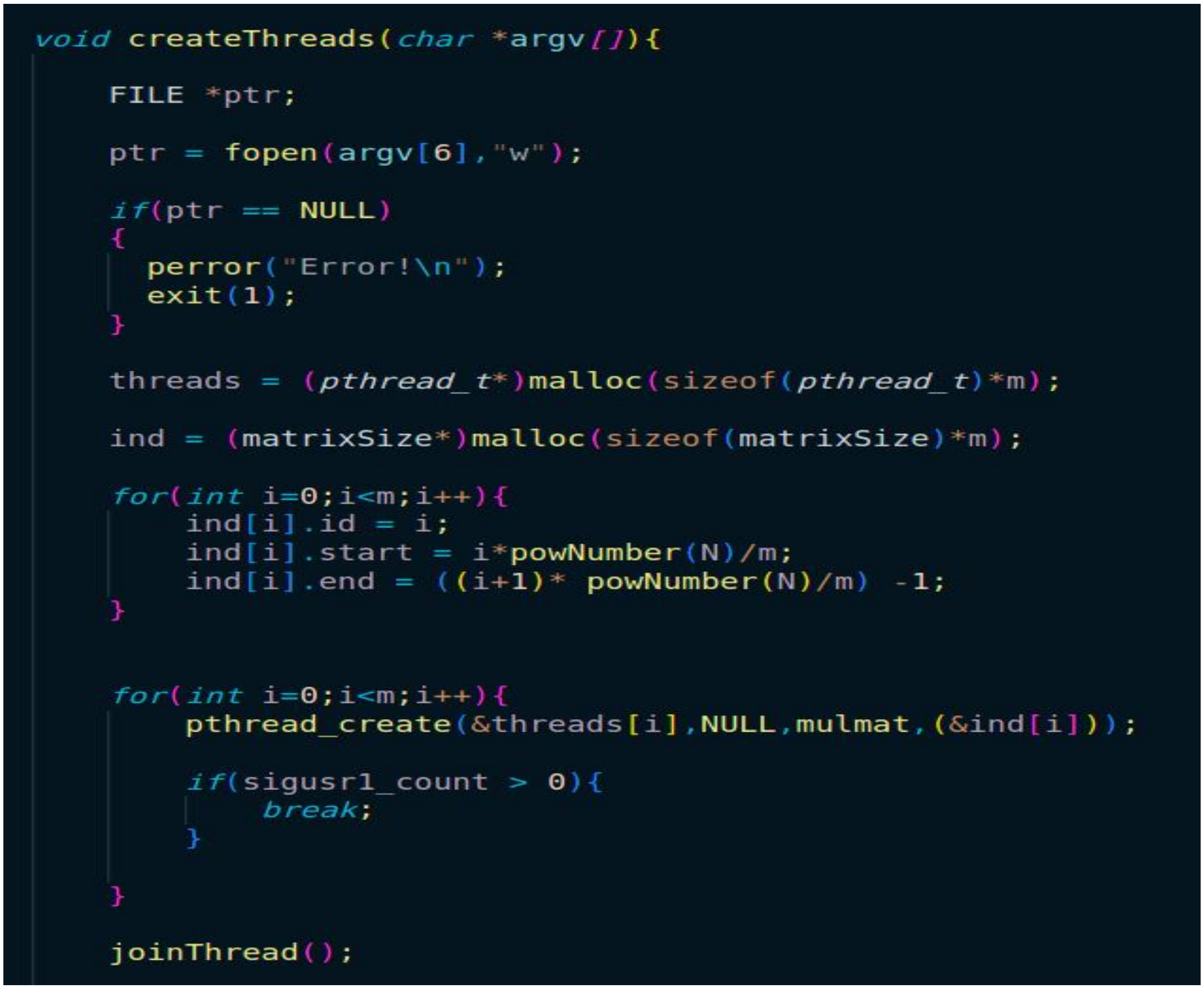
- In this function I made space for my double and single dimensional pointers with malloc. At the end of my main function, I released all of them with free. I call the assign() function where I will use the matrices.



```
int powNumber(int N){
    int result;
    result = pow(2,N);

    return result;
}
```

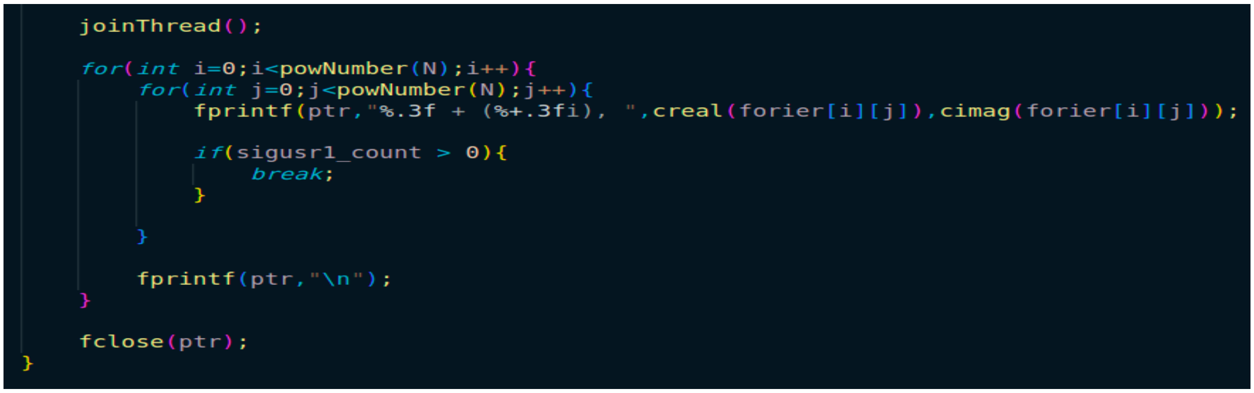
- In this function I get 2^n.

-createThreads():



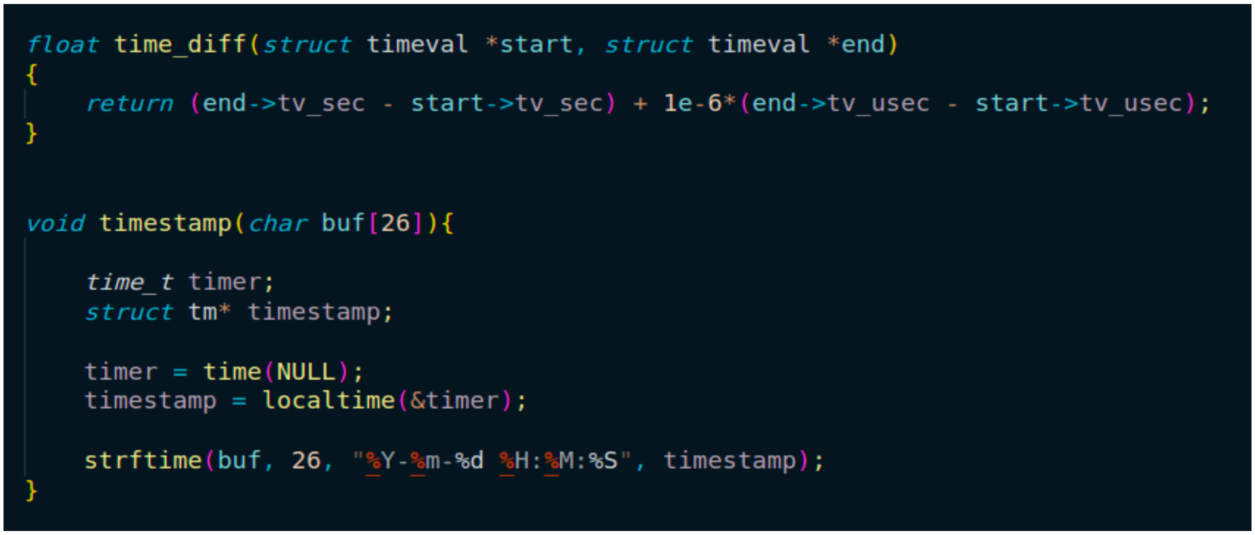
```
void createThreads(char *argv[]){

    FILE *ptr;

    ptr = fopen(argv[6],"w");

    if(ptr == NULL)
    {
      perror("Error!\n");
      exit(1);
    }

    threads = (pthread_t*)malloc(sizeof(pthread_t)*m);

    ind = (matrixSize*)malloc(sizeof(matrixSize)*m);

    for(int i=0;i<m;i++){
        ind[i].id = i;
        ind[i].start = i*powNumber(N)/m;
        ind[i].end = ((i+1)* powNumber(N)/m) -1;
    }


    for(int i=0;i<m;i++){
        pthread_create(&threads[i],NULL,mulmat,(&ind[i]));

        if(sigusr1_count > 0){
            break;
        }

    }

    joinThread();
```

- In this function I calculate how my threads will split the matrix. So here I do the multiplied form of the matrices, then join and make the barrier and call the other DFT function.



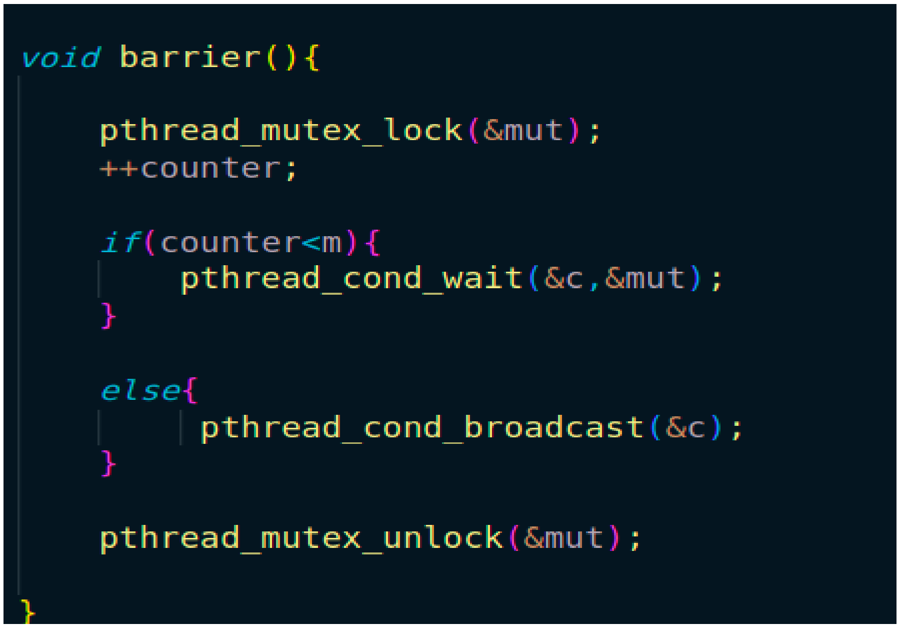
```
    joinThread();

    for(int i=0;i<powNumber(N);i++){
        for(int j=0;j<powNumber(N);j++){
            fprintf(ptr,"%.3f + (%+.3fi),  ",creal(forier[i][j]),cimag(forier[i][j]));

            if(sigusr1_count > 0){
                break;
            }

        }

        fprintf(ptr,"\n");
    }

    fclose(ptr);
}
```

- Thanks to these functions, I suppress the timestamp and how many seconds the threads do the job.

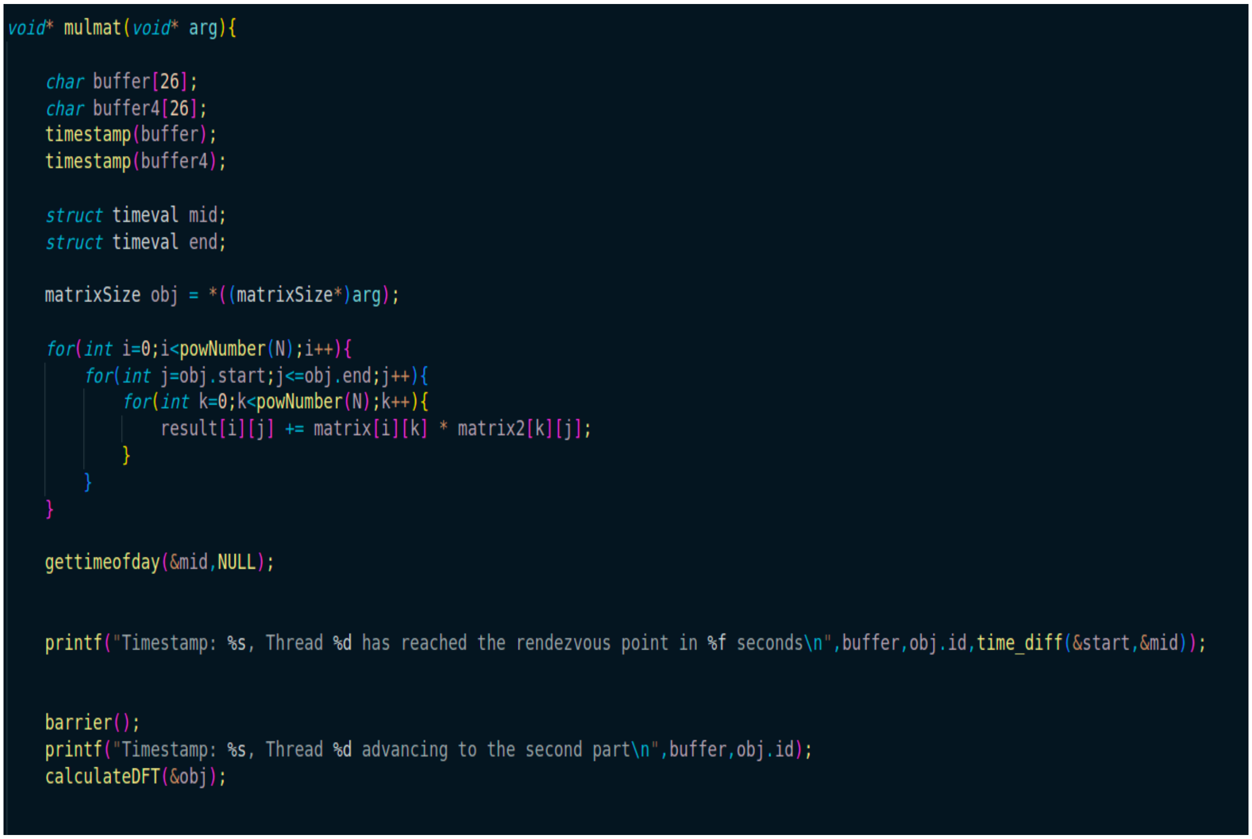


```
float time_diff(struct timeval *start, struct timeval *end)
{
    return (end->tv_sec - start->tv_sec) + 1e-6*(end->tv_usec - start->tv_usec);
}


void timestamp(char buf[26]){

    time_t timer;
    struct tm* timestamp;

    timer = time(NULL);
    timestamp = localtime(&timer);

    strftime(buf, 26, "%Y-%m-%d %H:%M:%S", timestamp);
}
```

- Here, after finding the result of the A*B matrices, I call the barrier function to avoid confusion and then calculate the DFT.



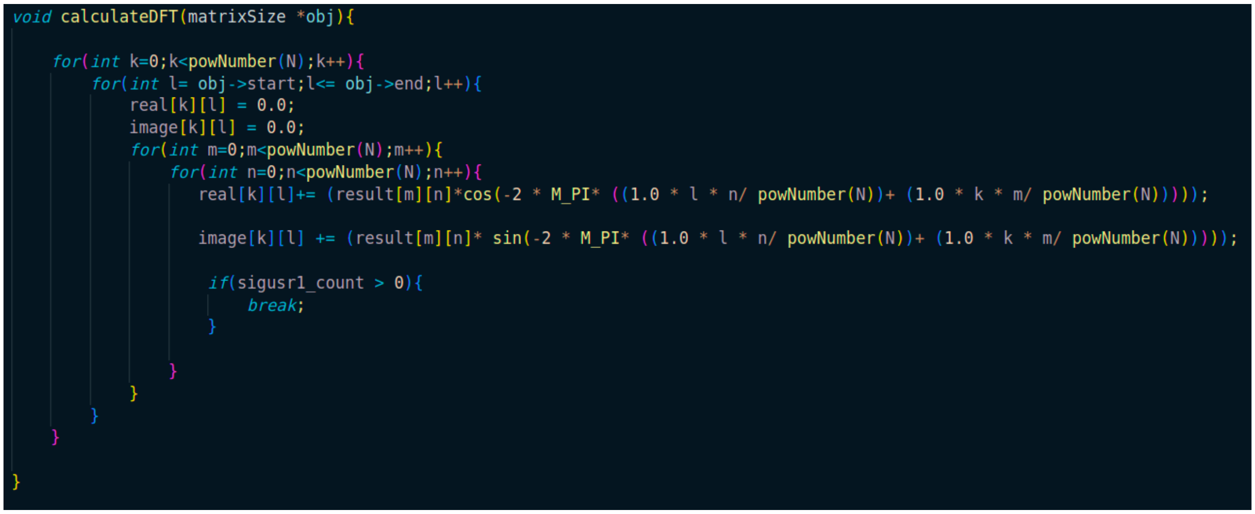
```
void barrier(){

    pthread_mutex_lock(&mut);
    ++counter;

    if(counter<m){
        pthread_cond_wait(&c,&mut);
    }

    else{
         pthread_cond_broadcast(&c);
    }

    pthread_mutex_unlock(&mut);

}
```



```c
void* mulmat(void* arg){

    char buffer[26];
    char buffer4[26];
    timestamp(buffer);
    timestamp(buffer4);

    struct timeval mid;
    struct timeval end;

    matrixSize obj = *((matrixSize*)arg);

    for(int i=0;i<powNumber(N);i++){
        for(int j=obj.start;j<=obj.end;j++){
            for(int k=0;k<powNumber(N);k++){
                result[i][j] += matrix[i][k] * matrix2[k][j];
            }
        }
    }

    gettimeofday(&mid,NULL);


    printf("Timestamp: %s, Thread %d has reached the rendezvous point in %f seconds\n",buffer,obj.id,time_diff(&start,&mid));


    barrier();
    printf("Timestamp: %s, Thread %d advancing to the second part\n",buffer,obj.id);
    calculateDFT(&obj);
```

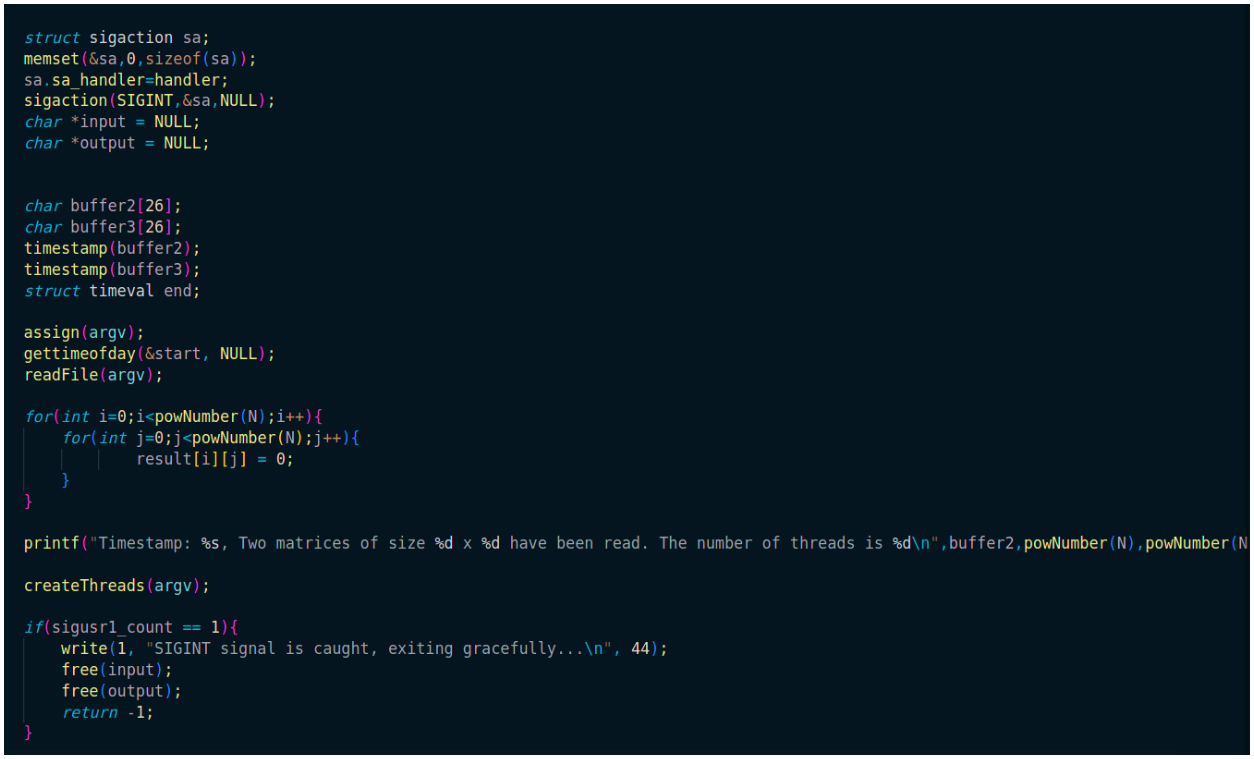
calculateDFT():

- I used 4 nested for loops to calculate the DFT formula. I have fulfilled the requirements of the formula. I applied the DFT formula using the result of the A*B matrices. And I use math library for calculation.

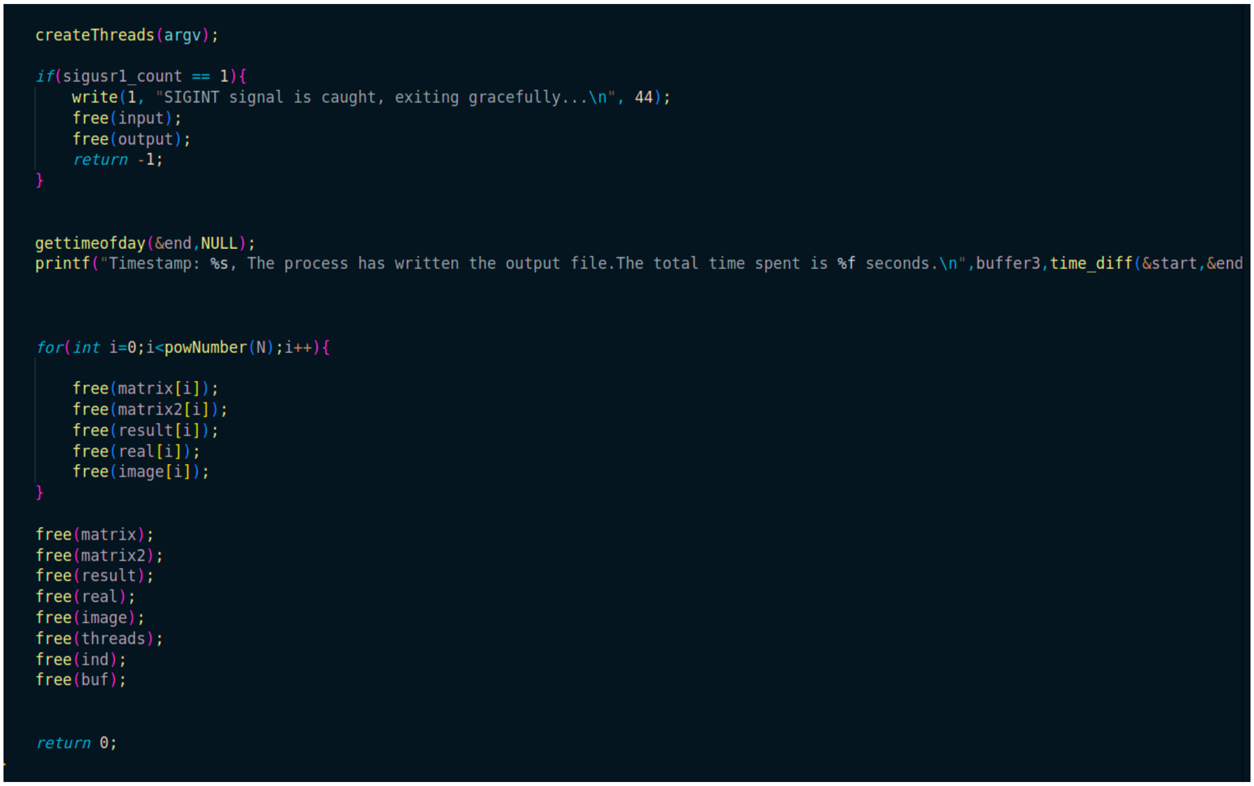


```c
void calculateDFT(matrixSize *obj){

    for(int k=0;k<powNumber(N);k++){
        for(int l= obj->start;l<= obj->end;l++){
            real[k][l] = 0.0;
            image[k][l] = 0.0;
            for(int m=0;m<powNumber(N);m++){
                for(int n=0;n<powNumber(N);n++){
                    real[k][l]+= (result[m][n]*cos(-2 * M_PI* ((1.0 * l * n/ powNumber(N))+ (1.0 * k * m/ powNumber(N)))));

                    image[k][l] += (result[m][n]* sin(-2 * M_PI* ((1.0 * l * n/ powNumber(N))+ (1.0 * k * m/ powNumber(N)))));

                     if(sigusr1_count > 0){
                         break;
                     }

                }
            }
        }
    }

}
```

Main():



```
struct sigaction sa;
memset(&sa,0,sizeof(sa));
sa.sa_handler=handler;
sigaction(SIGINT,&sa,NULL);
char *input = NULL;
char *output = NULL;


char buffer2[26];
char buffer3[26];
timestamp(buffer2);
timestamp(buffer3);
struct timeval end;

assign(argv);
gettimeofday(&start, NULL);
readFile(argv);

for(int i=0;i<powNumber(N);i++){
    for(int j=0;j<powNumber(N);j++){
        result[i][j] = 0;
    }
}

printf("Timestamp: %s, Two matrices of size %d x %d have been read. The number of threads is %d\n",buffer2,powNumber(N),powNumber(N

createThreads(argv);

if(sigusr1_count == 1){
    write(1, "SIGINT signal is caught, exiting gracefully...\n", 44);
    free(input);
    free(output);
    return -1;
}
```



```
createThreads(argv);

if(sigusr1_count == 1){
    write(1, "SIGINT signal is caught, exiting gracefully...\n", 44);
    free(input);
    free(output);
    return -1;
}


gettimeofday(&end,NULL);
printf("Timestamp: %s, The process has written the output file.The total time spent is %f seconds.\n",buffer3,time_diff(&start,&end



for(int i=0;i<powNumber(N);i++){

    free(matrix[i]);
    free(matrix2[i]);
    free(result[i]);
    free(real[i]);
    free(image[i]);
}

free(matrix);
free(matrix2);
free(result);
free(real);
free(image);
free(threads);
free(ind);
free(buf);


return 0;
```

- I call my functions on Main, I put my timestamp functions that I use while printing, I put the necessary parameters for SIGINT. Finally I made the places I reserved with malloc free.

MY DESIGN DECISION:

- I created a structure so that threads share the columns and defined 2 variables, start and end. I also kept another variable called id to keep the names of threads in that structure. I used my matrices that I will use multiplication and my matrix that I will assign its result to as a global value. First, I made space with malloc for the pointers I’m going to use. In my createThreads() function, I made space for my index and threads with malloc. Then I split the running threads according to the columns, then I called my function that finds the matrix multiplications.

WHICH REQUIREMENTS I ACHIEVE WHICH REQUIREMENTS I FAIL:

- I have fulfilled all the tasks completely and provided all the controls.