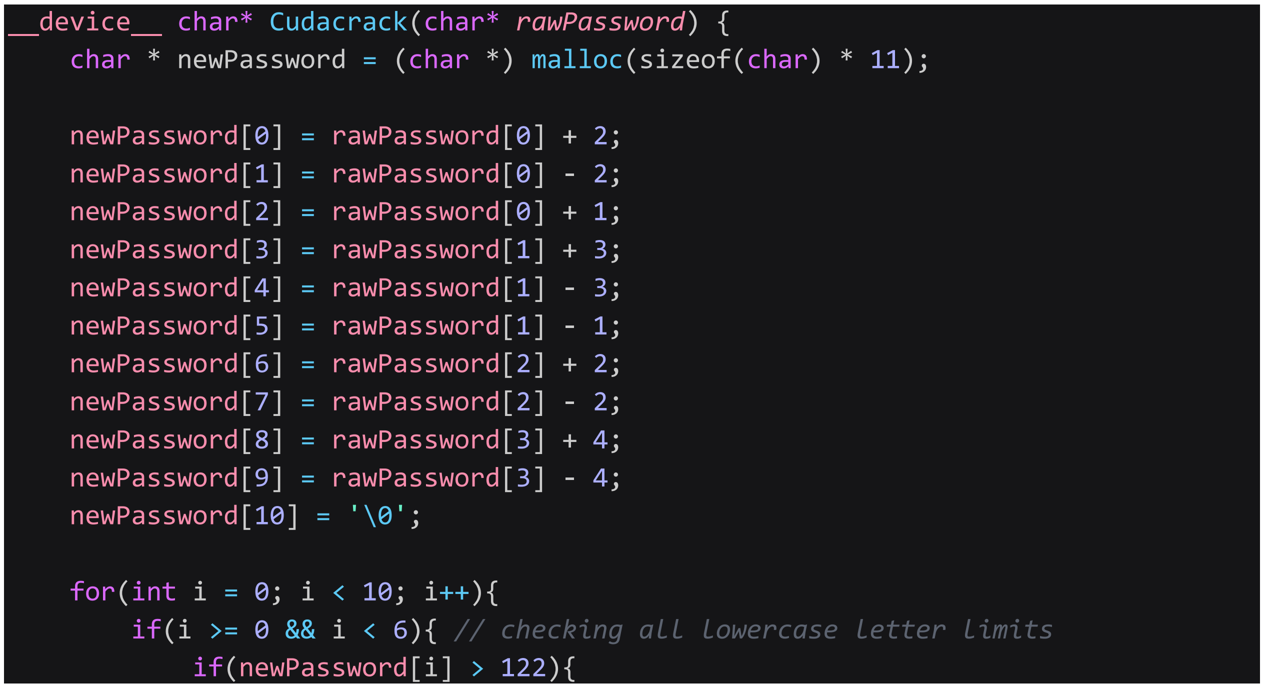
TASK 3:

This CUDA C program utilizes parallel processing on a GPU to crack an encrypted password. It employs a CUDA kernel, decryptPass, to generate potential password combinations and decrypt them using a function called Cudacrack. The code efficiently searches for the correct password by leveraging GPU threads. The main function initializes character sets, allocates memory on both the CPU and GPU, launches the kernel, and prints the results, displaying the original encrypted password and the decrypted password if found. The program demonstrates a parallelized approach to password decryption using CUDA.

1. **Generate encrypted password in the kernel function (using CudaCrypt function) to be compared to original encrypted password:**
   The Cudacrack function in the CUDA kernel is responsible for generating an encrypted password based on a given raw password. The transformation involves adding and subtracting specific values to each character in the raw password. The resulting encrypted password is then checked against the target encrypted password.



```
__device__ char* Cudacrack(char* rawPassword) {
    char * newPassword = (char *) malloc(sizeof(char) * 11);

    newPassword[0] = rawPassword[0] + 2;
    newPassword[1] = rawPassword[0] - 2;
    newPassword[2] = rawPassword[0] + 1;
    newPassword[3] = rawPassword[1] + 3;
    newPassword[4] = rawPassword[1] - 3;
    newPassword[5] = rawPassword[1] - 1;
    newPassword[6] = rawPassword[2] + 2;
    newPassword[7] = rawPassword[2] - 2;
    newPassword[8] = rawPassword[3] + 4;
    newPassword[9] = rawPassword[3] - 4;
    newPassword[10] = '\0';

    for(int i = 0; i < 10; i++){
        if(i >= 0 && i < 6){ // checking all lowercase letter limits
            if(newPassword[i] > 122){
                newPassword[i] = (newPassword[i] - 122) + 97;
            } else if(newPassword[i] < 97){
                newPassword[i] = (97 - newPassword[i]) + 97;
            }
        } else { // checking number section
            if(newPassword[i] > 57){
                newPassword[i] = (newPassword[i] - 57) + 48;
            } else if(newPassword[i] < 48){
                newPassword[i] = (48 - newPassword[i]) + 48;
            }
        }
    }
    return newPassword;
}
```

2. **Allocating the correct amount of memory on the GPU based on input data. Memory is freed once used:**
   Memory is allocated on the GPU for the alphabet, numbers, encrypted password, and the output password. The correct amount of memory is allocated based on the size of the data (alphabet, numbers, encrypted password, and output password).

```
// Allocate and copy memory for alphabet, numbers, encrypted password, and output
password on GPU
    char* gpuAlphabet;
    cudaMalloc((void**) &gpuAlphabet, sizeof(char) * 26);
    cudaMemcpy(gpuAlphabet, cpuAlphabet, sizeof(char) * 26,
cudaMemcpyHostToDevice);

    char* gpuNumbers;
    cudaMalloc((void**) &gpuNumbers, sizeof(char) * 10);
    cudaMemcpy(gpuNumbers, cpuNumbers, sizeof(char) * 10,
cudaMemcpyHostToDevice);

    char* gpuEncryptedPass;
    cudaMalloc((void**) &gpuEncryptedPass, sizeOfEncryptedPass);
    cudaMemcpy(gpuEncryptedPass, encryptedPass, sizeOfEncryptedPass,
cudaMemcpyHostToDevice);

    char* gpuOutputPass;
    cudaMalloc((void**) &gpuOutputPass, sizeOfEncryptedPass);
```

Memory is freed once it is no longer needed.

```
    // Free allocated GPU and CPU memory
    cudaFree(gpuAlphabet);
    cudaFree(gpuNumbers);
    cudaFree(gpuEncryptedPass);
    cudaFree(gpuOutputPass);
```

3. **Program works with multiple blocks and threads:**
   The CUDA kernel decryptPass is designed to work with multiple blocks and threads. The kernel is launched with a 2D grid (dim3(26, 26, 1)) and 2D block (dim3(10, 10, 1)) configuration to explore different combinations of alphabet and numbers.

```
// Launch CUDA kernel to decrypt the password
    decryptPass<<< dim3(26, 26, 1), dim3(10, 10, 1) >>>(gpuAlphabet, gpuNumbers,
gpuEncryptedPass, gpuOutputPass);
```

4. **Decrypted password sent back to the CPU and printed:**
   If a thread finds a matching password, it prints the result, including the encrypted and decrypted passwords. The decrypted password is sent back to the CPU, and the results are printed on the CPU.

```
    // Allocate and copy memory for the output password on CPU
    char* cpuOutputPass = (char*)malloc(sizeof(char) * 4);
    cudaMemcpy(cpuOutputPass, gpuOutputPass, sizeOfEncryptedPass,
cudaMemcpyDeviceToHost);

    printf("---\n");
    printf("Results:\n");

    // Print the decrypted and encrypted passwords or an error message
    if (cpuOutputPass != NULL && cpuOutputPass[0] != '\0') {
        printf("1. Given Encrypted Pass: '%s'\n", encryptedPass);
        printf("2. Found Decrypted Pass: '%s'\n", cpuOutputPass);
    } else {
        printf("Unable to determine a password.\n");
    }
```