David Morillo (1666540)
Adrià Muro (1665191)

1. Responeu les següents preguntes sobre el codi C proporcionat per la Pràctica

1. **Quines són les estructures de dades clau utilitzades en aquest codi?**

   Les estructures particulars d'aquest codi que considerem importants (trobats a la llibreria ***kmeanslib.h***) són:

   - **cluster**: Consta de:
     - **num_puntos**: Un enter de 32 bits (nombre total de punts en el clúster).
     - **r, g i b**: Tres enters de 8 bits. Utilitzat per a guardar la informació sobre el punt on es troba el centre del clúster (centroide) en l'espai on es troben totes les dades.
     - **media_r, media_g i media_b**: Tres enters de 8 bits. Tres variables on, es guarda la informació sobre la mitjana posicional de tots els punts del clúster (ens seveix per a posicionar el centroide a la pròxima iteració).
   - **rgb**: Consta de:
     - **b, g i r**. Tres enters de 8 bits. Utilitzats per a guardar la posició de cada punt en els 3 eixos (colors) en l'espai.
   - **image**: Consta de:
     - **length, width i height**: Enters de 32 bits. Mida de la imatge
     - **header**: Un array de enters de 8 bits de longitud de 54 elements
     - **pixels**: Un apuntador a l'estructura **"rgb"**.
     - **fp**: Un apuntador a l'estructura **"FILE"** (per a realitzar operacions d'entrada i sortida amb la imatge).

2. **Com es manegen els arxius d'entrada i sortida en aquest codi?**

   Els arxius d'entrada es gestionen a partir de la funció **read_file** que agafa la imatge original i la càrrega a memòria.

   El següent fragment de codi mostra com es llegeix l'arxiu d'entrada (mImage) que conté la imatge original. En cas d'error s'imprimeix un missatge conforme no s'ha pogut obrir i es surt (*exit(6)*).

```
int read_file(char *name, image* mImage)

{

    if((mImage->fp=fopen(name, "r")) == NULL) {

        printf("No fue posible abrir el BMP.\n");

        exit(6);
```

```
}
```

Pel que fa als arxius de sortida, es fa via la funció **write_file**, aquesta funció escriu una imatge de sortida, juntament amb els centroides de clústers trobats, a fitxers de sortida. Utilitza les següents crides a funcions d'entrada/sortida.

A més, es fan servir altres crides a funcions d'entrada/sortida com printf per mostrar missatges a l'usuari en cas d'errors o informació rellevant durant l'execució del programa.

3. **Com s'inicialitzen els centroides en la funció kmeans?**

```
// Init centroids
printf("STEP 2: Init centroids\n");
for(i = 0; i < k; i++)
{
    random_num = rand() % num_pixels;
    centroides[i].r = pixels[random_num].r;
    centroides[i].g = pixels[random_num].g;
    centroides[i].b = pixels[random_num].b;
}
```

Podem veure que la funció **kmeans** inicia determinant les coordenades (r, g, b) de cada un dels centroides els quals volem tenir un clúster (k clústers, k centroides). Això ho fa amb una funció generadora de nombres pseudoaleatoris d'entre 0 i 1 i ho multiplica pel nombre de píxels.

4. **Quin propòsit té el càlcul del checksum en la funció *getChecksum*?**

El càlcul del checksum proporciona una manera de comprovar si els centroides o l'actualització de les dades d'aquest, una vegada passen per l'algoritme k-means, s'han modificat de manera errònia. Si el valor resultant de la funció getChecksum canvia, pot ser indici d'un error en el codi que s'ha de resoldre.

5. **Com es gestiona l'assignació i alliberament de memòria per als píxels i centroides?**

L'assignació de memòria per als píxels es fa de la següent manera:

```
mImage->length = (mImage->width * mImage->height);
```

```
mImage->pixels  = malloc(mImage->length * sizeof(rgb));
```

Primerament es guarda el nombre total de píxels que conté la imatge multiplicant l'amplada i altura de la imatge. Una vegada ja es té la quantitat total de píxels (mImage->length) s'utilitza la funció *malloc* que permet assignar dinàmicament memòria. L'espai total necessari és el resultat de multiplicar la quantitat total de píxels amb l'espai que ocupa cadascun (sizeof(rgb)).

L'alliberament de memòria dels píxels és molt simple. S'utilitza la funció free(mImage.pixels) al final del "main" que allibera l'espai de memòria assignat als píxels.

Pel que fa a l'assignació de memòria dels centroides es du a terme a partir d'aquesta comanda:

```
centroids = malloc(k * sizeof(cluster));
```

En el cas de la variable "centroids" que és la variable que guarda el retorn del punter de la funció malloc, guarda el nombre de bytes de la mida de l'estructura de clúster multiplicada pel nombre de centroides que posem d'entrada.

Per alliberar la memòria destinada als centroides es fa ús de la funció free(centroids).

En el cas dels centroides i els píxels si es dóna el cas que no hi ha memòria suficient, es farà una impressió que no n'hi ha i la funció retornarà el valor NULL .

6. Com es mesura i mostra el temps d'execució de l'algoritme K-Means?

Per dur a terme el càlcul del temps d'execució de l'algoritme K-Means s'utilitza la funció *gettimeofday*. Primerament s'agafa el temps actual en milisegons i es guarda a la variable *start* (&start, 0). Seguidament s'executa l'algoritme K-Means i una vegada finalitza, es torna a agafar el temps en aquell moment i es guarda (&end, 0). Per obtenir el temps d'execució es resten els temps finals amb els inicials guardats a les variables. Si es dóna el cas que el final (tv_usec) esdevé negatiu es suma un 1 segon i es resta el mateix a (tv_sec) perquè el resultat total no es vegi afectat.

```
end.tv_sec = end.tv_sec - start.tv_sec;

end.tv_usec = end.tv_usec - start.tv_usec;

if (end.tv_usec < 0) {  end.tv_usec += 1000000; end.tv_sec--; }
```

```
printf("\nTime to Kmeans is %ld seconds and %ld microseconds\nChecksum value = %u\n", end.tv_sec, end.tv_usec, checksum);
```

7. **Com gestiona el codi situacions d'error?**

   La gestió de les situacions d'error inclou la comprovació dels valors que retornen funcions determinants, com les de memòria o lectura/escriptura d'arxius i els missatges d'error adaptats al tipus d'error detectat durant l'execució del codi.

   En el cas de **kmeanslib.c** s'observa que tant la funció **read_file** com **write_file** tenen impressions d'errors depenent del cas. Durant el procés de manipulació d'arxius poden sorgir molts tipus d'errors i és important saber de quin tipus es tracta, així com si es produeix durant la lectura, escriptura o tancament d'aquests.

   Possibles Errors:

   read_file: "No fue posible abrir el BMP", "Error fseek", "Error Writing File", "Error Reading H File", etc.

   write_file: "Cannot create output file", "Error closing file", "Clusters.txt cannot be created", etc.

   Tanmateix, també hi ha comprovacions entorn la memòria:

```
centroids = malloc(k * sizeof(cluster));

if(centroids == NULL){

    printf("Not enough memory.\n");

    return -4;

}
```

   En aquest exemple, una vegada s'ha reservat memòria per guardar els centroides, es comprova si hi ha suficient memòria. En el cas que no hi hagi prou memòria s'imprimeix un missatge d'error ("Not enough memory") i es retorna el valor "-4" (negatiu = error).

8. **Com es generen els arxius de sortida i quina informació contenen?**

   En la funció **write_file** es crea l'arxiu *clusters.txt* que guarda els pixels RGB de cada centroide que hem donat com a paràmetre inicial (k).

```
if((fp=fopen("clusters.txt", "w")) == NULL){

            printf("Clusters.txt cannot be created. Do you have space |
owner in current directory?\n");

            return 1;

        }
```

   De la mateixa manera, es crea un arxiu BMP que conté la imatge inicial segmentada on cada píxel s'ha assignat al centroide més proper.

```
        if((fp=fopen(name, "w")) == NULL){

            printf("Cannot create output file, do you have write permissions
for current directory?.\n");

            return 1;

        }

        if(fwrite(mImage->header, sizeof(uint8_t), sizeof(mImage->header), fp)
!= sizeof(mImage->header)){

            printf("Error writing BMP.\n");

            return 1;

        }

        for(i = 0; i < mImage->width; i++){

            for(j = 0; j < mImage->height; j++){

                closest = find_closest_centroid(&mImage->pixels[i *
mImage->height + j], centroids, k);

                pixel.r = centroids[closest].r;

                pixel.g = centroids[closest].g;

                pixel.b = centroids[closest].b;
```

```
            if(fwrite(&pixel, sizeof(uint8_t), 3, fp) != 3){
                printf("Error writing BMP.\n");
                return 1;
            }
        }
    }
```

En aquest codi s'obre l'arxiu de sortida BMP per escriptura, s'itera per cada píxel de la imatge original, s'aplica la funció *find_closest_centroid* que s'assigna a un centroide, s'escriuen els valors RGB a l'arxiu de sortida BMP i es tanca.