Image.cpp :

1] Image::Image() : width(0), height(0), buffer(NULL) {}

Αρχικά βάζουμε το Image:: για να έχουμε πρόσβαση σε μεθόδους και μεταβλητές της κλάσης Image. Σε αυτόν τον constructor αρχικοποιώ τις τιμές των width και height , 0 και του buffer μου null δηλαδή κενό με 0 στοιχεία. Δημιουργεί ένα κενό αντικείμενο Image.

2] Image::Image(unsigned int width, unsigned int height) : width(width), height(height), buffer(NULL)

Αρχικοποιούμε τις μεταβλητές width και height. Φτιάχνει μία εικόνα της οποίας έχουμε μόνο μήκος και πλάτος και οχι τα πίξελ.

3] Image::Image(unsigned int width, unsigned int height, const Color data_ptr)*
{
Image(width, height); → καλώ την παραπάνω μέθοδο να αναθέσει τιμές
setData(data_ptr); → μεταφέρω τα δεδομένα από το data_ptr στον buffer μου
}

Παίρνει ως όρισμα μία άλλη είκονα και δημιουργεί αντικείμενο Image με τα στοιχεία αυτής της εικόνας.

4] Image::Image(const Image& src)
{
width = src.width;
height = src.height;
*buffer = new Color[width * height];*
*memcpy(buffer, src.buffer, width * height * sizeof(Color)); //tha mporousame kai pio apla na kanoume buffer = src.buffer*
}

Είναι width*height * Color διότι το μέγεθος του αντικειμένου είναι width*Heigth και είναι τύπου Color άρα επί το sizeof(Color).

Αυτός είναι ο λεγόμενος copy constructor:
Αν δεν τον ορίσουμε, ο μεταγλωττιστής τον παράγει αυτόματα – Δημιουργεί ένα νέο αντικείμενο που να είναι αντίγραφο ενός άλλου, ίδιου τύπου – Το πρόβλημα είναι ότι το τί σημαίνει «αντίγραφο» δε μπορεί να το γνωρίζει ο μεταγλωττιστής πάντα. Έτσι λοιπόν :
Δημιουργώ ένα αντικείμενο Image με βάση τα στοιχεία ενός άλλου αντικειμένου Image .Θέτω ως width και height , τη τιμή που μας δείχνει η αναφορά του ήδη υπάρχοντος αντικειμένου στο width και το height αυτού.Τέλος δημιουργώ ένα αντικείμενου τύπου Color και αντιγράφω τα στοιχεία του αντικειμένου στο καινούργιο αντικείμενο με την εντολή memcpy , την οποία εξηγώ παρακάτω πιο αναλυτικά αλλά μία απλή εξήγηση είναι : Συνάρτηση C αντιγραφής μπλοκ μνήμης. Χρήσιμη για γρήγορη αντιγραφή συνεχούς περιοχής δεδομένων.

5] Image::~Image()
{
if (buffer)
delete[] buffer;
}

Καταστροφέας.*Οταν έχω δυναμικό πίνακα ή pointer πάντα πρέπει να τον καλώ ώστε να αποδεσμεύεται μνήμη και να μην κινδυνεύσω σε περίπτωση διαρροής μνήμης.*

6] Image& Image::operator=(const Image& right)

```
{
    if (this == &right) {
        return *this;
    }
   //deallovate memory
    if (buffer != NULL) {
        delete[] buffer;
    }

    width = right.width;
    height = right.height;
    //diethese kainourgio xwro
    buffer = new Color[width * height];
    //antegrapse ta stoixeia
    memcpy(buffer, right.buffer, width * height * sizeof(Color));

    return *this; //i alliws *this = right
}
```

Αυτός είναι ο λεγόμενος copy assignment operator

Δύο αντικείμενα της ίδιας κλάσης μπορούν να έχουν πρόσβαση το ένα στα μη δημόσια πεδία του άλλου.Ο τελεστής ανάθεσης (=) επιχειρεί να δώσει την τιμή ενός δεδομένου σε ένα άλλο . Όταν και οι 2 τελεσταίοι είναι του ίδιου τύπου, τότε αυτός ο τελεστής αποτελεί μια από τις ειδικές μεθόδους μιας κλάσεις, τον copy assignment operator (τελεστής ανάθεσης με αντιγραφή). Όπως με τον κατασκευαστή αντιγραφής, έτσι και στην περίπτωση του =, ο μεταγλωττιστής δημιουργεί έναν έτοιμο τελεστή =, αν δε δώσουμε το δικό μας.Καλείται όταν έχει δημιουργηθεί ένα αντικείμενο και γίνεται σε αυτό ανάθεση άλλου αντικειμένου ίδιου τύπου.Άρα λοιπόν αντίστοιχα εδώ με αντικείμενα Image αρχικά κοιτάω τις δύο διευθύνσεις αν είναι ίδιες , αν είναι σημαίνει ότι έχει γίνει αυτοανάθεση άρα δεν κάνουμε τίποτα .Αποδεσμεύω την μνήμη για να μην κινδυνεύσω πάλι από διαρροή μνήμης και μετά λειτουργώ όπως κι στον Copy constructor.

7] void Image::setPixel(unsigned int x, unsigned int y, Color& value)

```
{
    if (x > width || y > height) {
        std::cout << "The cordinates are out of bounds" << std::endl;
        return;
    }
    else {
        int pos = x + (y * width);
        buffer[pos] = value;
    }
```

*Ορίζει τις τιμές RGB για κάθε ένα pixel .Έχω αρχικά έναν έλεγχο ώστε να δω αν οι διαστάσεις που μου δίνει είναι μεγαλύτερες από το width και το heigth αν είναι τότε επιστρέφω και τυπώνω ότι είναι εκτός ορίων.Αλλιώς τοποθετώ στον πίνακα buffer στη θέση x + (y*width) την καίνούργια τιμή*

(χρώμα).Αυτή η συνάρτηση προκύπτει διότι έστω ότι θέλω το 4ο pixel σε ένα δυσδιάστατο πίνακα 6x6 θα βρίσκεται στην θέση χ = 4 και y =0 άρα pos = 4.

8] Color Image::getPixel(unsigned int x, unsigned int y) const

```
{
    if (x > width || y > height) { // an i eikona einai 1000 x 1000 kai zitisoume to pixel x kai y
megalitera einai out of bounds
        return imaging::Color(0, 0, 0); //an oi sintetagmenes ksepernane to width kai to heigth
                                        //epestrepse mauro
    }
    else {
        int pos = x + (y * width);  //koita set pixel eksigisi
        return buffer[pos];
    }
}
```

Γυρνάει το χρώμα της εικόνας στο συγκεκριμένο x,y.Κάνω πάλι τον ίδιο έλενχο και αν ισχύει γυρνάει μαύρο , αλλιώς το χρώμα του συγκεκριμένου πιξελ από το πίνακα buffer .

9]

```
void Image::setData(const Color*& data_ptr) //xrisimeuei opote thelei na allaksw ola ta dedomena
tis eikonas
{
    if (buffer == NULL) { //ama einai adeios perase ta xaraktiristika tis eikonas ppm
        buffer = new imaging::Color[width * height];
    }

    memcpy(buffer, data_ptr, width * height * sizeof(Color));
```

Αντιγράφει τα δεδομένα εικόνας από ένα εξωτερικό raw buffer στο εσωτερικό buffer εικόνας.Αν είναι άδειος (buffer==NULL → έχω null γτ στα λινουξ δν μαναγνωριζοταν το nullptr) περνάω τα χαρακτηριστικά της εικόνας αλλιώς αντιγράφω όλα τα στοιχεία από τον data_ptr στον buffer.

10] bool Image::load(const std::string& filename, const std::string& format)

```
{

    std::string nameCopy = filename.substr(filename.size() - 3,3);
    nameCopy.c_str();
    format.c_str();
    for (int i = 0; i <3; i++){
      tolower(nameCopy[i]);
      tolower(format[i]);
    }


    if ( format == nameCopy) {
        int w;
```

Filename: η συμβολοσειρά του αρχείου για να διαβάσει τα δεδομένα . Αποθηκεύω σε μία μεταβλητή String τα τρεία τελευταία ψηφία της συμβολοσειράς(δηλαδή το ppm).

Αποθηκεύω την παραπάνω μεταβλητή σε ένα πίνακα String , το ίδιο και το format το οποίο καθορίζει τη μορφή αρχείου σύμφωνα με την οποία τα δεδομένα εικόνας θα πρέπει να αποκωδικοποιηθούν από το αρχείο.Τα κάνω και τα δύο να είναι μικρά ώστε να μην επηρεάζεται αν θα είναι Ppm ή Ppm κτλ. Και ύστερα συγκρίνω αν είναι ίδια , αν δεν είναι τυπώνω ότι δεν είναι ο τύπος αρχείου που θέλω.

Καλώ την readppm θ αεξηγηθεί πιο κάτω και την αποθηκεύω σε ένα πίνακα τύπου float αφού αυτό επιστρέφει.

```
    int h;
    float* ImageData = ReadPPM(filename.c_str(), &w, &h);

    if (ImageData == NULL) {      → αν δεν έδωσε τίποτα
       std::cout << "Error reading ppm file" << std::endl;
       return false;
    }

    this->buffer = new Color[w * h];
    this->width = w;
    this->height = h;

    for (size_t i = 0; i<(w)*(h)*3; i+=3){
      for ( int j = 0; j <(w)*(h); j++){
        this->buffer[j].r = ImageData[i] ;
         this->buffer[j].g =  ImageData[i+1];
         this->buffer[j].b = ImageData[i+2];
      }
    }
    // ή αλλιώς με την παρακάτω εντολή :
    //memcpy(this->buffer, ImageData, 3 * w * h * sizeof(float));
    return true;
  }
  else {
    std::cout << "Not a ppm file!" << std::endl;
    return false;
  }
}
```

Είναι το this που αναφέρεται στο width του αντικειμένου, δλδ της Image.

Αν η μεταβλητή μας είναι δείκτης σε αντικείμενο μπορώ αντί για (*p1).print να βάλω p1->print()

*Φορτώνει τα δεδομένα της εικόνας . Ο πίνακας ImageData είναι τύπου floats και περιέχει διαδοχικές τιμές των rgb που διαβάστηκαν από το αρχείο ενώ ο buffer περιέχει αντικείμενα Color δηλαδή τα pixels που έχουν ως περιεχόμενα (μεταβλητές) τα rgb.Γιαυτό στο πρώτο for έχουμε ότι αυξάνεται κατά 3 το ι kai πάει μέχρι 3*width*height ενώ στο δεύτερο width*heigth που είναι οι διαστάσεις του buffer.*

11]

```
bool Image::save(const std::string& filename, const std::string& format)
 {


    if (this->buffer == NULL) {
       std::cout << "Cannot save null buffer" << std::endl;
       return false;
    }
    //einai 3 * width*height logw twn rgb pou einai 3 byte
    float* ImData = new float[3 * width * height];

     for (size_t i = 0; i<(*width)*(*height)*3; i+=3){
       for ( int j = 0; j <(*width)*(*height) j++)
```

```
        ImData[j] = this->buffer[i].r;
        ImData[j+1] = this->buffer[i].g;
        ImData[j+2] = this->buffer[i].b;
        j+=3;
       }
    //memcpy(ImData, this->buffer, width * height * sizeof(Color));

    return WritePPM(ImData, width, height, filename.c_str());
  }
```

Ουσιαστικά είναι σαν να κάνει το αντίστροφο από την load , αφού γράφει τα περιεχόμενα του file.Αν είναι άδειος ο buffer τότε τυπώνω μήνυμα και γυρνάω false , αλλιώς αντιγράφω αντίστροφα τα στοιχεία από το buffer μου στον ImData .

ppm.cpp:

```
12] float* ReadPPM(const char* filename, int* w, int* h)
  {
    std::fstream file(filename, std::ios::in | std::ios::binary);
    if (!file.is_open()) {
      return NULL;
    }

    std::string str;
    file >> str;
    if (str != "P6") {
      std::cout << "This is not a P6 file!" << std::endl;
      return NULL;
    }

    int width, height, pixels;
    file >> width >> height >> pixels;
    *w = width;
    *h = height;

    if (w == NULL || width <= 0 || h == NULL || height <= 0) {
      std::cout << "Error! Width and Height of the image are not valid!" << std::endl;
      return NULL;
    }
    if (pixels > 255 || pixels <= 0) {
      std::cout << "Error! Max value of pixels is not valid!" << std::endl;
      return NULL;
    }

    //to seekg: Sets the position where the next character is to be inserted into the output stream
    //to cur  : current position within sequence
    //to kanoume gia na paralipsei  ti teliki grammi i to keno sto telos tis kefalidas
    file.seekg(1, std::ios::cur);
    //pinaka 3 byte-> 1 pixel
    unsigned char pix[3];
```

Ανοίγουμε ένα αρχείο μέσω ρεύματος εισόδου ή εξόδου αρχικοποιώντας ένα αντικείμενο τύπου Fstream.To *std::ios::in | std::ios::binary σημαίνει ότι ανοίγει το αρχείο για ανάγνωση και ως δυαδικό αντί για αρχείο κειμένου.*

Έλεγχος για άνοιγμα αρχείου ,θα γραφόταν fstream::is_open όμως έχουμε κάνει include<fstream> οπότε είμαστε καλλημένοι.

Διαβάζει το πρώτο string , αν δν είναι P6 τότε δεν είναι δεκτό.

Διαβάζει τα επόμενα τρεία int που αντιπροσωπεύουν το width το heigth και pixels αντίστοιχα .Κι παρακάτω κάνει τους απαραίτητους ελέγχους.

Αφού τα rgb μπορούν να πάρουν τις τιμές 0 , 1 ,2.

Η εντολή seekg θέτει την θέση του επόμενου χαρακτήρα που θέλουμε να εισαχθεί στο output stream .Το ένα συμβολίζει τα byte και το *std::ios::cur* την τρέχουσα θέση συν τα bytes δηλαδή εδώ συμβαίνει το εξής: Έχουμε διαβάσει τελευταίο τα pixels , όμως σε κάθε εικόνα από το 255 μέχρι τα αλαμπουρνέζικα είτε έχει ένα κενό είτε μία γραμμή κενή άρα πάμε και λέμε τοποθέτησου μία θέση (1 byte) από εκεί που βρίσκεσαι , έτσι πάει στο πρώτο αλαμπουρνέζικο και είναι έτοιμο να αρχίσει να τα διαβάζει.

Τα διβάζω ως char γιατί είναι ένα byte και με βολεύει στο τρόπο που θέλω να τα διαβάζω.

```
float* ppm_array = new float[3 * width * height];

int j = 0;
for (int i = 0; i < width * height; ++i) {
   file.read((char*)pix, 3);

   ppm_array[j] = int(pix[0]) / 255.f;
   ppm_array[j + 1] = int(pix[1]) / 255.f;
   ppm_array[j + 2] = int(pix[2]) / 255.f;

   j += 3;
}

return ppm_array;
}
```

Διαβάζω κάθε πιξελ ή αλλιώς 3 byte ένα ένα ως χαρακτήρες.

Τα κάνουμε int και ύστερα τα μετατρέπουμε σε float διότι θέλουμε :
Η Image εσωτερικά να χρησιμοποιεί δεδομένα σε μορφή float. Κατά την ανάγνωση, να μετατρέπονται οι τιμές από το διάστημα ακεραίων [0,255] στο κανονικοποιημένο διάστημα αριθμών κινητής υποδιαστολής [0.0,1.0].

Η ReadPPM επιστρέφει ένα δείκτη σε μια τύπου float array διάστασης 3 w *h*, η οποία περιέχει τα δεδομένα της εικόνας.*

```
12] bool WritePPM(const float* data, int w, int h, const char* filename)
{
   if (data == NULL) {
      return false;
   }
   else {
      std::fstream file(filename, std::ios::out | std::ios::binary);
      file << "P6" << std::endl
           << w << " "
           << h << std::endl
           << "255" << std::endl;

      unsigned char r, g, b;
      int j = 0;
      for (int i = 0; i < w * h; i++) {
         r = data[j] * 255;
         g = data[j+1] * 255;
         b = data[j+2] * 255;

         file << r << g << b;
         j += 3;
      }
      file.close();
   }
```

Δηλώνω ότι θα γράψω κιόλας

Παίρνω κάθε πιξελ της εικόνας κι το φέρνω σε μορφή byte.

Γράφει ένα buffer εικόνας ως αρχείο PPM.Το data περιέχει το buffer της εικόνας. Τα δεδομένα είναι ταξινομημένα με τριπλέτες RGB με τιμές στην περιοχή [0,1].

main.cpp:

```
Image getNegative(const Image& input)
{
   Image pic = input;

   for (int i = 0; i < pic.getWidth(); i++) {
      for (int j = 0; j < pic.getHeight(); j++) {
         Color c = pic.getPixel(i, j);
         Color new_col(1., 1., 1.);
         new_col = new_col - c; → ektelese ti sinartisi p(1,1,1)-p(x,y)
         pic.setPixel(i, j, new_col);
      }
   }

   return pic;
}
int main(int argc , char * argv[]){

    string filename;
    if ( argv[1] != NULL){
      filename = argv[1];
    }else{
      cout << "File name of the Image to load: " << endl;
      cin >> filename;
    }

    fstream file(filename.c_str() , ios:: out | ios:: in | ios::binary);

    if ( !file.is_open()){
      cerr << "File could not be opened!" << endl;
    }


    Image img;
    if ( img.load(filename , "ppm")){
      cout << "Image dimensions are : " << img.getWidth() << " X " << img.getHeight() << endl;
      Image neg = getNegative(img);
      size_t pos = filename.find(".");
      string filename2 = filename;
      filename2.insert(pos  , "_neg" );
      neg.save(filename2 , "ppm");
      return 0;
    }
```

Δημιουργεί το αρνητικό της εικόνας. Ουσιαστικά παίρνει το κάθε πιξελ με την εντολή getPixel που αντιστοιχεί στις συγκεκριμένες συντεταγμένες , το αφαιρεί από το (1,1,1) δηλαδή το άσπρο (γιαυτό υπάρχουν κι τα + /- δηλ ολα τα ενδιάμεσα χρώματα από το άσπρο και το μαύρο) κι ύστερα το επανατοποθετεί με το καινούργιο χρώμα.

Εντολές ώστε το ανρητικό να αποθηκεύεται σε _neg.

Βιβλιοθήκη και Makefile σε λινουξ:

Το Makefile(ορίζεται με targets , εμείς εδώ έχουμε πολύ λίγα αρχεία οπότε θα έχουμε μόνο ένα που το ονομάζουμε all:) δεν είναι παρά ενα απλό script γραμμένο στο mousepad ,ουσιαστικά μόλις το καλείς αυτό εκτελεί όλες τις εντολές που έχεις εκεί μία προς μία. Δηλαδή το δικό μας επειδή θέλουμε να έχουμε βιβλιοθήκη μόνο της ppm θα έχουμε τις παρακάτω εντολές:

g++ -c ppm.cpp : κάνει compile το αρχείο ppm.cpp (το αρχείο της βιβλιοθήκης) θα δημιουργήσει ένα αρχείο ppm.o που είναι το αντικείμενο που έχει μεταγγλωτιστεί.

ar rs libppm.a ppm.o : δημιουργεί την βιβλιοθήκη για την ppm

ar: δημιουργεί ένα archive αρχείο
r: σημαίνει αντικατάσταση ή δημιουργία νέου αρχείου
s : σημαίνει ότι θα δημιουργήσει ένα ευρετήριο των περιεχομένων του αρχείου. Αυτό απαιτείται από τον μεταγλωττιστή.
Libppm : όνομα βιβλιοθήκης
ppm.o : συμπεριλάβουμε τα .o αρχεία σε μια βιβλιοθήκη

g++ main.cpp Image.cpp -L . -lppm -o neg : compilation του κύριου προγράμματος

g ++ source_files_separated_by_space
L : σημαίνει να προσθέσετε τον κατάλογο στη διαδρομή lib
-l {library_name} : βρείτε στο lib path lib {library_name} .a ή lib {library_name} .so και στη συνέχεια να συνδεθείτε σε αυτό
-o output_name : το όνομα του εκτελέσιμου