Proiektua: 1. Fasea

Laburpena

Konputagailuen bidezko Grafikoak irakasgaian, *OpenGL* eta C lengoaia erabiliz, programa bat garatu behar da. Programaren garapena lau fasetan banatzen da; dokumentu honek lehenengo fasearen nondik norakoak jasotzen ditu. Tutorial hau burutzen duzunean praktikaren azken hiru faseei aurre egiteko prest egongo zara.

1 C lengoaia

Programaren egitura aurkeztu aurretik, praktika burutzeko beharrezkoak izango diren C lengoaiaren zenbait kontzeptu berrikusiko ditugu.

1.1 Erakusleak (pointerrak)

Pointerren kontzeptua C eta C++ lengoaien oinarrizko kontzepturik garrantzitsuena eta ahaltsuena da, memoria zuzenean atzitzea ahalbideratzen baitu. Memoria zuzenean erabiltzea interesgarria bezain arriskutsua da eta, hortaz, pointerrak arreta handiz erabili behar dira.

Pointerrek ez dute edozein balio gordetzen, memoria-helbide bat baizik. Demagun `int` motako aldagai bat daukagula, `var` izenekoa. Aldagai hori erazagutzean memorian zati bat erreserbatzen da bere balioa gordetzeko. Demagun memoria zati horren helbidea 1001 dela. C eta C++ lengoaietan memoria zati horren helbidea pointer batean, `ptr`, gordetzeko aukera daukagu. Pointerrak berak ere bere helbidea izango du (2047, esate baterako), eta bere barruan `var` aldagaiaren helbidea, hau da, 1001, gordeko du. 1 irudian eskematxo bat ikus daiteke.

Ikus dezagun C lengoaian nola erazagutu aldagaiak eta pointerrak:

```
int var = 50;
int *ptr = &var;
```

Lehendabiziko lerroan `var` aldagaia erazagutzen da, 50 balioa esleituz. Bigarren lerroan, berriz, pointer bat erazagutu dugu. Sintaxia hauxe da `<pointer mota> *<izena>`. Ikus daitekeenez, pointer bat erazagutzeko zein motako aldagaien pointerra izango den adierazi behar da. Adibidean `int` motakoa da, baina edozein oinarrizko edo definitutako motakoa (`char`, `float`, `double`, ...) izan daiteke. Erazagutu dugun pointerrean, `var` aldagaiaren helbidea gorde nahi dugu eta, horretarako, `&var` sintaxia erabili behar da, aldagaiaren helbidea lortzeko.

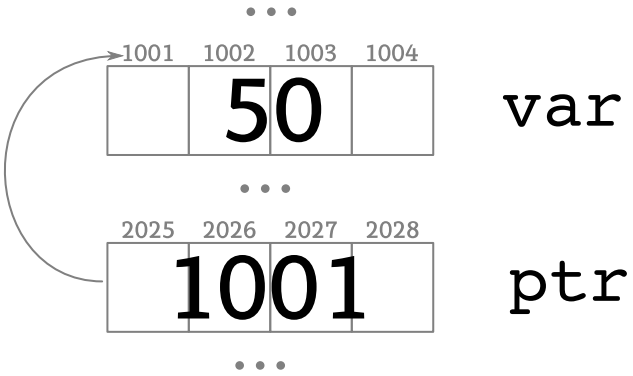
Pointerrak memoria helbidetzat hartzen dira, eta erazagutzean balio konkretu bat ez badiegu ematen ere, balio bat (hau da helbide bat) hartuko dute. Helbide hori zorizkoa izango da. Hori dela eta, hasieratu gabeko pointer bat baldin badugu, erabiltzen saiatzen bagara zoirzko memoria zati horretan dagoena atzituko da eta edozer gerta daiteke (ohikoena atzitu ezin dugun zati baten helbidea izatea da eta, beraz, atzitzen saiatzean `segmentation fault` errorea jasoko dugu). Hori dela eta, *oso garrantzitsua da pointer guztiak hasieratzea balio batekin edo, hartu behar duen balioa jakin ezean, 0 balioarekin.*

Pointerrak, barnean gordetzen duen helbidean dagoena atzitzeko erabil daitezke. Azter dezagun kode zati hau:

```
float var, var2;
float *ptr = 0;
var = 50.5;
```

Ikasturtea: 3. Maila
Irakaslea: Borja Calvo
Telefonoa: 943 01 50 13
E-posta: borja.calvo@ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)



1 irudia: Memoriaren eskema. Irudian ikus daitekeen bezala, `int` motako aldagaiek 4 byte tamaina dute. Helbideen tamaina ere, 32bit-eko sistemetan 4 byte da (64bit-eko sistemetan 8 izango da, byte guztien helbideratzeko).

```
ptr = &var;
var2 = *ptr;
```

Bi `float` aldagai eta pointer bat erazagutzen ditugu. Aldagaiak ez dira hasieratzen baina pointerra bai (0 balioarekin). Hirugarren lerroan `var` aldagaiari 50.5 balioa esleitzen diogu eta, ondoren, `ptr` pointerrean aldagai honen helbidea gordetzen dugu. Azken lerroan `var2` aldagaian `var` aldagaian dagoena gordetzen dugu, pointerra erabiliz. Kodean ikus daitekeenez, pointerrak duen helbidean dagoen balioa atzitzeko `*ptr` sintaxia erabili behar dugu.

2 Egituren eta moten definizioa

C lengoaia objektuei bideratuta ez egon arren, badu mekanismo bat datu egiturak definitzeko. Demagun gure kontaktuak kudeatzeko programa bat garatu nahi dugula. Kontaktu bakoitzeko izena eta telefonoa gorde nahi badugu, ondoko datu egitura erabil dezakegu:

```
struct contact {
  char *name;
  char *telefone;
};
```

`char * name` sintaxiarekin, gero ikusiko dugun moduan, karaktere bektore bat erazagutu dezakegu.[1] C lengoaian, beraz, egiturak definitzeko `struct` hitz erreserbatua erabiltzen da. Behin egitura definiturik, aldagai berri bat erazagutzeko `struct contact aldagaiaren_izena` idatzi beharko dugu; pointerrak ere erazagutu daitezke, `struct contact *aldagaiaren_izena` kodea erabiliz. Egitura definituta dagoen bezala, erazagupenetan `struct` hitz erreserbatua erabiltzea derrigorrezkoa da. Hori saihes daiteke aldagai mota berri bat definituz, `typedef` hitz erreserbatua erabiliz:

```
struct contact {
  char *name;
  char *telefone;
};
typedef struct contact contact;
```

Mota berria definitzen badugu, aldagaiak eta pointerrak erazagutzeko sintaxia sinplifikatzen da: `contact aldagaiaren_izena` eta `contact *aldagaiaren_izena`, hain zuzen.

[1] Izan ere, mota guztietako bektoreak pointerren bidez definitzen dira

3 Programa sinple bat

C lengoaian programa bat idazteko `main` funtzio bat behar dugu, programa exekutatzean funtzio horri deituko baitzaio. Adibide honetan, `main` funtzioaz gain aurreko atalean ikusi dugun egitura (hedatuta) erabiltzen da:

```
#include <stdio.h>
#include <malloc.h>

struct contact{
  char *name;
  char *telefone;
  struct contact *next;
};
typedef struct contact contact;

int main(int argc, char** argv) {
  int option = 0;
  char *name = malloc(sizeof (char)*64);
  char *tlf = malloc(sizeof (char)*16);
  contact *list_ptr = 0;
  contact *list_aux_ptr = 0;
  while (option!=3)
  {
    printf("Zer egin nahi duzu?\n\t1.- Telefono berri bat sartu\n\t2.- Telefonoak ikusi\n\t3.- Amaitu\n\n");
    scanf("%d", &option);
    switch(option){
      case 1:
        printf("Sartu izena:");
        scanf("%s",name);
        printf("Sartu telefonoa:");
        scanf("%s",tlf);
        list_aux_ptr = malloc(sizeof(contact));
        list_aux_ptr->name = name;
        list_aux_ptr->telefone = tlf;
        list_aux_ptr->next=list_ptr;
        list_ptr = list_aux_ptr;
      break;
      case 2:
        list_aux_ptr = list_ptr;
        while(list_aux_ptr!=0)
        {
          printf("%s: %s\n",list_aux_ptr->name,list_aux_ptr->telefone);
          list_aux_ptr = list_aux_ptr->next;
        }
        printf("\n");
      break;
      default:
      break;
    }
  }
}
```

Azter dezagun adibidea. Lehenengo bi lerroetan `#include` komandoak daude. Komando horiek, programa garatzeko behar ditugun liburutegiak kargatzeko erabiltzen dira (Javaren `import` komandoaren baliokidea). Adibide honetan bi liburutegi behar dira, bat sarrera/irteera erabiltzeko eta bestea memoria erreserbatzeko.

Ondoren, kontaktuak gordetzeko erabiliko dugun egitura daukagu; aurreko atalekoari eremu berri bat gehitu diogu, kontaktu zerrenda bat era sinplean inplementatzeko.

Gero funtzio bakarra dago: `main`[2]. Funtzioaren hasieran erabiliko ditugun aldagai guztien erazagutzen ditugu. Hori, Javarekin alderatuta, C lengoaiaren ezaugarri garrantzitsua da; *erabiltzen diren aldagai guztiak funtzioaren hasieran erazagutu behar dira, kodea hasi aurretik.*

Lehendabiziko lerroan `int` motako aldagai bat erazagutzen da. Aldagai horretan erabiltzaileak adierazitako aukera gordeko da. Ondoren bi `char` motako pointer erazagutzen dira, izena eta telefonoa gordetzeko. C lengoaian bektore bat erazagutzeko pointerrak erabiltzen dira [3].

Lerro horretan `malloc` funtzioa erabiltzen da. Funtzio hori memoriako zati bat erreserbatzeko (*Memory ALLOCation*) erabiltzen da, eta parametro bakarra dauka: zenbat memoria erreserbatu behar den. Sistemaren

[2] Funtzio honen parametroetan `char** argv` dago, pointer baten pointer bat ...

[3] Aurrerago ikusiko dugunez, pointerrak lehendabiziko elementoaren helbidea gordeko du, eta hurrengoak atzitzeko lehendabizikoaren helbideari kopuru ezagun bat gehituko diogu.

arabera behar den tokia alda daitekeenez, zehazki zenbat `byte` behar diren jakiteko beste funtzio interesgarri bat daukagu, `sizeof`. Funtzio honek mota/egitura bakoitzak behar duen memoria itzultzen du.

Karaktere bektore bat erazagutu nahi dugunez, zein tamainakoa izango den jakin behar dugu. Lehenengo kasuan tamaina 64 izango da eta bigarrenean, berriz, 16. Hori dela eta, 64 eta 16 karaktere gordetzeko memoria beharko dugu eta horregatik, memoria tamaina osoa kalkulatzeko, `sizeof` funtzioak bueltatzen duenari 64 edo 16 biderkatu behar diogu.

`malloc` funtzioak memoria erreserbatu ondoren zati horren lehenengo `byte`aren helbidea itzultzen du. Kodean helbide hori pointer batean (`char *name`) jasotzen dugu.

Erazagupenekin amaitzeko bi `contact` egitura pointer sortzen ditugu, bat gure kontaktuen zerrenda gordetzeko eta bestea zerrenda hori korritzeko. Hasieran ez dugu ezer ere ez izango pointer horieta eta, horregatik, *zero balioarekin hasieratzen ditugu.*

Erazagupen guztien ostean `while` bakarra dago[4]; `option` aldagaia 3 izan arte begiztaren barruan dagoen kodea exekutatuko da. `while`ko lehendabiziko lerroan `printf` komandoa daukagu, pantailan mezuak inprimatzeko erabiltzen dena. Kasu honetan komandoaren sintaxia oso sinplea da, baina konplexuagoa izan daiteke, gero ikusiko dugun bezala.

Bigarren lerroan `scanf` funtzioa dago. Funtzio hori teklatutik informazioa jasotzeko erabiltzen da. Funtzioak, kasu horretan, bi parametro ditu. Lehenengoa `string` bat da, `"%d"`, eta bigarrena `option` aldagaiaren helbidea. `"%d"` kodeak funtzioak zeinudun zenbaki oso bat irakurriko duela esan nahi du; irakurritako zenbakia `option` aldagaian gordeko da. Funtzioak zein aldagai mota irakurri behar duen jakiteko kodeak erabiltzen dira. Ondoko taulak beste kode batzuk jasotzen ditu:

Kodea	Formatua
%c	Karaktere bat (`char`)
%s	Karaktere bektore bat (`char*`)
%d	Zeinudun zenbaki osoa, notazio dezimalean
%i	Zeinudun zenbaki osoa
%u	Zeinu gabeko zenbaki osoa
%f	Koma higikorra
%e	Notazio zientifikoa, berretzaile 'e' hizkia erabiliz
%E	Notazio zientifikoa, berretzaile 'E' hizkia erabiliz

Bi komando hauen ostean, `switch-case` egitura bat daukagu. Bertan, bi aukera ditugu (`default` kasuan ez baita ezer egiten). Lehendabiziko kasuan berriro `printf` eta `scanf` funtzioak erabiltzen dira. Funtzio horien bidez erabiltzaileari izena eta telefonoa eskatzen zaizkio. Jasotako informazioa `name` eta `tlf` bektoreetan gordetzen dugu. Ondoren, `contact` motako egitura batentzat memoria erreserbatzen da eta bertan jasotako informazioa gordetzen dugu.

Kontaktu zerrenda bat lortzeko, sortu dugun kontaktu berriaren `next` eremuan orain arte geneukan kontaktu lista gordetzen dugu, eta uneko zerrendari apuntatzen dion pointerrean sortu dugun egituraren helbidea gordetzen dugu.

Bigarren aukeran kontaktu guztiak inprimatzen dira. Horretarako, uneko kontaktutik abiatuta informazioa pantailaratu eta hurrengo kontaktura pasatu behar gara; `next` eremuan zero balioa topatzen dugunean kontaktu zerrenda amaitu dela jakingo dugu. Informazioa inprimatzen duen lerroa aztertzen badugu, goiko taulan dauden kodeak mezuak pantailatzerako nola erabil daitezkeen ikusiko dugu. `printf` funtzioaren lehenengo parametroa `"%s:  %s\n\n"` da. Honek esan nahi du `string` bat inprimatuko dela, gero bi puntu, gero hutsune bat, beste `string` bat eta, amaitzeko, bi lerro-jauzi; bi `string` horiek bigarren eta hirugarren parametroekin (hau da, `name` eta `tlf`) pasatzen zaionak izango dira.

3.0.1 Programak konpilatzen

Aurreko atalean ikusi dugun programa sinplea (linux motako sistematan) konpilatzeko, *main.c* izeneko fitxategi batean gorde eta terminalean ondoko kodea exekutatu behar dugu:

[4] Aspektu honetan C eta Java lengoaien sintaxia oso antzerakoa da.

```
gcc -o programa main.c
```

Goiko komandoak `gcc` konpiladorea erabiltzen du, linux sistematan datorrena delako, baina beste hainbat konpiladore erabil daitezke. Lerro hau exekutatu ondoren, *programa* izeneko fitxategi exekutagarria sortuko da.

4 Praktikaren deskribapena

Irakasgaian garatu behar den praktikaren helburua 3D grafikoak kudeatzeko eta bistaratzeko aplikazio bat programatzea da. Garapen prozesua lau fasetan banatuta dago. Lehenengoan objektuak fitxategitatik irakurri eta bistaratu egingo dugu, bigarren fasean objektuak aldatzeko aukera gehituko diogu gure programari, hirugarrenean kameraren kudeaketari ekingo diogu eta, amaitzeko, laugarren fasean argien kudeaketa landuko dugu. Erabiltzailearekin behar den elkarrekintza guztia teklatuaren bidez izango da.

Praktikan sortu behar den programa garatzeko hiru liburutegi erabiliko ditugu. Lehendabizikoa `GL` da, zeinek *OpenGL*ko oinarrizko eragiketak eta funtzioak eskaintzen dizkigun. Horrez gain, oinarrizko funtzioak hedatzen dituen `GLU` liburutegia ere erabiliko dugu. Amaitzeko,`glut` liburutegia erabiliko dugu, objektuak leiho batean bistarateko.

Tutorial honetan lehenengo fasea pausoz pauso aztertuko dugu, *OpenGL*ren zenbait kontzeptu aztertuko ditugu eta programaren oinarrizko egitura azalduko dugu.

5 Aplikazioaren hasierako fitxategiak

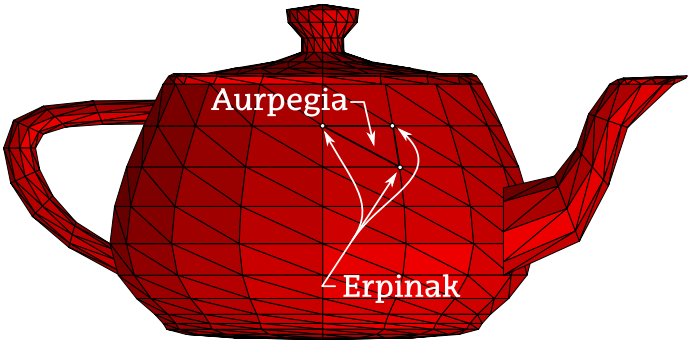
C tutorialean ikusi dugun adibidea oso txikia denez, kode guztia fitxategi bakar batean geneukan. Praktikaren kasuan, garatu behar duzuen programa askoz ere konplexuagoa da eta *kodea fitxategitan egituratzea gomendagarria da*. Hona hemen hasierako fasean erabiltzen diren fitxategiak:

- *main.c* - Fitxategi horretan `main` funtzioa izango dugu. Funtzio horrek sistema hasieratuko du eta begizta nagusia martxan jarriko du. Horrez gain, fitxategi honetan zenbait aldagai ere erazagutzen dira.
- *io.c* - Lehen esan bezala, programa kontrolatzeko teklatua erabiliko dugu. Fitxategi honek elkarrekintza hori ahalbideratzeko funtzioak jasoko ditu.
- *io.h* - *io.c* fitxategian dauden zenbait funtzio beste fitxategietatik eskuragarri izan behar direnez, goiburu-fitxategi honetan funtzio horiek erazagutuko ditugu.
- *display.c* - Objektuak bistaratzeko, kamera kontrolatzeko eta argien eta materialen erabiltzeko behar diren funtzioak fitxategi honetan bilduko ditugu.
- *display.h* - *display.c* fitxategiari dagokion goiburu-fitxategia.
- *definitions.h* - Funtzioak inplementatzeko zenbait egitura beharko ditugu. Egitura guzti horiek fitxategi honetan erazagutuko ditugu. Horrez gain, zenbait konstante erabiliko ditugu programaren zehar, kodea ulergarriagoa izan dadin. Konstante horien definizioak ere fitxategi honetan jasoko ditugu.

Jarraian, zerrendatu ditugun fitxategietan dagoen kodearen aspektu garrantzitsuenak aztertuko ditugu.

5.1 *definitions.h* fitxategia

Fitxategi honek bi atal ditu. Alde batetik, fitxategiaren hasieran zenbait definizio ditugu. Erabiltzen ditugun konstante guztiak hemen jasota daude, aldaketarik egin nahi izanez gero erraz topatzeko. Definizio guztien ostean programan erabiliko ditugun datu egiturak ditugu. Hurrengo ataletan egitura horiek laburki aztertuko ditugu.



2 irudia: Aurpegiez osaturiko 3D objektu baten adibidea.

5.1.1 Puntuak, bektoreak eta koloreak

Gure objektuak deskribatzeko 3D puntuak eta bektoreak beharko ditugu (ikusi 2 irudia). Elementu horiek, ondoko egituren bidez adieraziko ditugu:

```
typedef struct {
    GLdouble x, y, z;
} point3;

typedef struct {
    GLdouble x, y, z;
} vector3;
```

Nahiko objektu sinpleak dira, puntuaren edo bektorearen hiru koordenatuak besterik ez gordetzen baitituzte. Koordenatuak koma higikorrako zenbakiak dira, baina C lengoaiaren `double` aldagai mota erabili beharrean, *OpenGL*k definitutako `GLdouble` mota erabiliko dugu[5].

Koloreak adierazteko egitura antzerakoa da, `color3`, baina koordenatuak gorde beharrean, hiru balioetan gorriaren (*red*, r), berdearen (*green*, g) eta urdinaren (*blue*, b) intentsitateak gordeko ditugu; hau da, koloreak RGB adierazpidearen bidez kodetuko ditugu.

```
typedef struct {
    GLfloat r, g, b;
} color3;
```

5.1.2 Erpinak, aurpegiak eta objektuak

Objektuak aurpegien bidez eraikita daude eta, horiek, erpinen bidez adierazita, 2 irudian ikus daiteken legez. Beraz, bi egitura erabiliko ditugu, `vertex`, erpinak adierazteko eta `face`, aurpegiak edo poligonoak adierazteko.

```
typedef struct {
    point3 coord;
    GLint num_faces;
} vertex;

typedef struct {
    GLint num_vertices;
    GLint *vertex_table;
} face;
```

Aurpegiek erpinak parteka ditzaketenez, zenbait kasutan uneko erpina zenbat aurpegitan dagoen jakitea oso erabilgarria izan daiteke. Hori dela eta, `vertex` egituran erpinaren koordenatuaz gain, erpin hori zenbat aurpegitan dagoen ere gordetzen da.

Aurpegiei dagozkienez, erpin zerrenda bat besterik ez dira. Edonola ere, koordenatu zerrenda moduan adierazteak arazo bat dakar: informazioa errepikatzen da. Erpin bat bi aurpegitan badago, bi aurpegi horiek informazioa gordeko dute. Hori saihesteko, aurpegiek erpinaren informazioa gorde beharrean, erpinaren indizea

[5] Era berean `GLint` eta `GLfloat` erabiliko ditugu

gordetzen dute; indize bakoitzari zein erpin dagokion jakiteko objektuaren egituran dagoen bektore bat erabiliko dugu. Beraz, `vertex_table` pointerrak lehenengo erpinaren indizeari apuntatuko dio. Bektorea korritzeko zenbat erpin dauden jakin behar dugunez, informazio hori `num_vertices` zenbaki osoan gordeko dugu.

Amaitzeko, objektuak gordetzeko erabiliko dugun egitura daukagu, `object3d`

```
struct object3d{
    GLint num_vertices;
    vertex *vertex_table;
    GLint num_faces;
    face *face_table;
    point3 min;
    point3 max;
    struct object3d *next;
};
typedef struct object3d object3d;
```

Egitura honetan zenbait elemento ditugu. Lehenik eta behin, erpinen eta aurpegien zerrenda izango dugu. Gogoratu `face` egituran dagoen erpin zerrenda erpinen indizeena dela; hau da, erpinaren koordenatuak berreskuratzeko objektuaren erpin-taula beharko dugu. Informazio honez gain, bi 3D puntu ditugu, `min` eta `max`. Puntu hauek objektuaren *mugak* adierazteko erabiliko ditugu; bi puntu horiek koordenatu bakoitzaren balio minimoa eta maximoa gordeko dituzte, hurrenez hurren.

Egituraren azken eremua, `next`, objektu zerrendak inplementatzeko da (zerrenda estekatua nola inplementatu ikusteko, ikus ezazu 1 atala).

5.2 *main.c* fitxategia

Fitxategi honetan (eta beste *.c gainontzeko guztietan) daukagun lehendabiziko gauza `include`-ak dira:

```
#include <stdio.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include "display.h"
#include "io.h"
#include "definitions.h"
```

Bi `include` mota ditugu. Lehenengo laurak sistemako liburutegienak dira, sarrera-irteera kontrolatzeko liburutegia eta aurreko atalean aipatutako *OpenGL*koak; azken hirurak guk sortutakoak dira.

Ondoren, aldagai globalak erazagutzen dira:

```
GLdouble _window_ratio;

GLdouble _ortho_x_min,_ortho_x_max;
GLdouble _ortho_y_min ,_ortho_y_max;
GLdouble _ortho_z_min,_ortho_z_max;

object3d * _first_object = 0;
object3d * _selected_object = 0;
```

Lehenengo aldagaiak leihoaren proportzioak gordeko ditu, objektuak bistaratzean deformatu ez daitezen. Hurrengo sei aldagaiak (`_ortho` aurrizkiarekin hasten direnak), bistaratzen edo proiekzio- eremuaren mugak definitzeko dira. Horien ostean bi `object3d` pointer ditugu, lehenengoak gure objektu-zerrenda gordetzeko eta bigarren uneoro zein objektu dagoen hautatuta jakiteko.

Aldagai global guztiak behin erazaguturik, `main` funtzioan erabiltzen den `initalization` funtzioa daukagu. Funtzio horrek bi hasieraketa motak ditu. Alde batetik, gure aldagai globalei balio lehenetsiak esleitzen dizkie; bestetik, *OpenGL*ko bi hasieraketa egiten ditu:

```
glClearColor(KG_COL_BACK_R, KG_COL_BACK_G, KG_COL_BACK_B, KG_COL_BACK_A);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

Lehenengo funtzioak pantaila ezabatzean erabiliko den kolorea ezartzen du (pantailaren atzeko planoaren kolorea, alegia), bere RGB osagaiak erabiliz.[6] Balio hauek guk (*definitions.h* fitxategian) definitutakoak dira.

[6] Izan ere, RGBA, azken balioa, alpha, kolorearen *gardentasuna* adierazten baitu.

Bigarren funtzioak, ostera, objektuen poligonoak nola marraztuko diren definitzen du. Kasu honetan erabiltzen diren konstanteak *OpenGL* liburutegian definitutakoak dira. Guk definitutako konstanteei erraz antzemateko, guztiei `KG` aurrizkia jarriko diegu, eta hitzak banatzeko _ sinboloa erabiliko dugu (7. atalak praktikan erabili behar diren kodetze-arauak azaltzen ditu).

Amaitzeko `main` funtzioa bera daukagu. Funtzio honetan, lehenengo aginduak teklen erabilerari buruzko informazioa pantailaratzen du (funtzio hau *io.c* fitxategian dago). Ondoren, jarraian dauden aginduak ditugu:

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGB);
glutInitWindowSize(KG_WINDOW_WIDTH, KG_WINDOW_HEIGHT);
glutInitWindowPosition(KG_WINDOW_X, KG_WINDOW_Y);
glutCreateWindow(KG_WINDOW_TITLE);
```

Funtzio guzti hauek erabiliko dugun leihoa hasieratzeko eta sortzeko erabiltzen dira. Besteak beste, leihoaren tamaina ta kokapena eta bere titulua ezartzen dituzte. Informazio guzti hori, ikus daitekeenez, guk definitutako konstantetan daukagu.

Hurrengo hiru funtzioek *callback* funtzio nagusiak ezartzen dituzte, hauxe baita `glut` liburutegiak leihoaren elkarrekintza kudeatzeko erabiltzen duen metodoa:

```
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
```

Lehenengo lerroaren eraginez, leihoa marraztu behar denean `display` funtzioari deituko zaio. Era berean, leihoaren tamaina aldatzean eta tekla bat sakatzean `reshape` eta `keyboard` funtzioei deituko zaie, hurrenez hurren.

Amaitzeko, lehen ikusi dugun hasieraketa burutzen da, `initialization` funtzioari deituz eta begizta nagusia martxan jartzen da, `glutMainLoop` funtzioari esker. Behin martxan, leihoa ikusiko dugu eta teklatutik bidaltzen ditugun aginduen zain gelditzen da.

5.3 *io.c*

Lehen aipatu bezala, erabiltzailearekin elkarrekintza teklatuaren bidez izango da. Horretarako, aurreko atalean `glut` liburutegiari tekla bat sakatzen denean `keyboard` funtzioa exekutatu behar dela esan diogu; funtzio horren kodea *io.c* fitxategian dago. Funtzio horrez gain, programaren erabilera pantailaratzen duen funtzioa ere *io.c* fitxategian aurki dezakegu.

Fitxategiaren hasieran ondoko kodea daukagu:

```
extern object3d * _first_object;
extern object3d * _selected_object;

extern GLdouble _ortho_x_min,_ortho_x_max;
extern GLdouble _ortho_y_min,_ortho_y_max;
extern GLdouble _ortho_z_min,_ortho_z_max;
```

Goiko kodean agertzen diren aldagaiak `keyboard` funtziotik atzigarriak izan behar dira, baina *main.c* fitxategian erazagututa daude. Hori dela eta, beste fitxategi batean daudela adierazteko `extern` hitz erreserbatua erabiltzen dugu.

Ondoren, kodean `print_help` funtzioa daukagu, zeinek ondoko informazioa pantailaratzen duen:

```
KbG Irakasgaiaren Praktika. Programa honek 3D objektuak
aldatzen eta bistaratzen ditu.

Egilea: Borja Calvo (borja.calvo@ehu.es)
Data: Irailak, 2014

FUNTZIO NAGUSIAK
<?>             Laguntza hau bistaratu
<ESC>           Programatik irten
<F>             Objektua bat kargatu
<TAB>           Kargaturiko objektuen artean bat hautatu
```

```
<DEL>           Hautatutako objektua ezabatu
<CTRL + ->      Bistaratze-eremua handitu
<CTRL + +>      Bistaratze-eremua txikitu
```

Laguntza honetan teklatuaren erabilera jasotzen da. Ikus daitekeenez, zenbait aukera ditugu; aukera guztiak kudeatzeko, `keyboard` funtzioan `switch-case` egitura handi bat izango dugu. Datozen ataletan egitura horretan dauden kasurik garrantzitsuenak.

5.3.1 Objektuak kargatu

Erabiltzaileak f edo F sakatzen duenean sistemak kargatu nahi dugun fitxategiaren bide-izena eskatu behar du. Ondoren, fitxategian dagoen informazioa `object3d` egitura batean gordeko da. Akatsen bat egonez gero, mezu bat pantailaratuko da; akatsik egon ezean, irakurritako objektua zerrendan sartuko da. Hona hemen kodea:

```
case 'F':
case 'f':
  printf("%s", KG_MSSG_SELECT_FILE);
  scanf("%s", fname);

  auxiliar_object = (object3d *) malloc(sizeof (object3d));
  read = read_wavefront(fname, auxiliar_object);

  switch (read) {
    case 1:
      printf("%s: %s\n", fname, KG_MSSG_FILENOTFOUND);
      break;
    case 2:
          printf("%s: %s\n", fname, KG_MSSG_INVALIDFILE);
            break;
        case 3:
          printf("%s: %s\n", fname, KG_MSSG_EMPTYFILE);
          break;
        case 0:
          auxiliar_object->next = _first_object;
          _first_object = auxiliar_object;
          _selected_object = _first_object;
          printf("%s\n",KG_MSSG_FILEREAD);
          break;
    }
    break;
```

Lehenik eta behin, mezu bat pantailaratzen da (mezu hori, beste guztiak bezala, `definitions.h` fitxategian definituta dago); ondoren fitxategiaren izen-bidea teklatutik jasotzen da, `fname` aldagaian. Fitxategia irakurri aurretik, *objektuaren informazioa gordetzeko memoria erreserbatu behar da.* Behin memoria erreserbatuta, `read_wavefront` funtzioa erabiltzen da fitxategia kargatzeko. Eraginkortasuna dela eta, funtzio honi objektu osoa pasatu beharrean, bere pointerra pasatu beharko diogu.

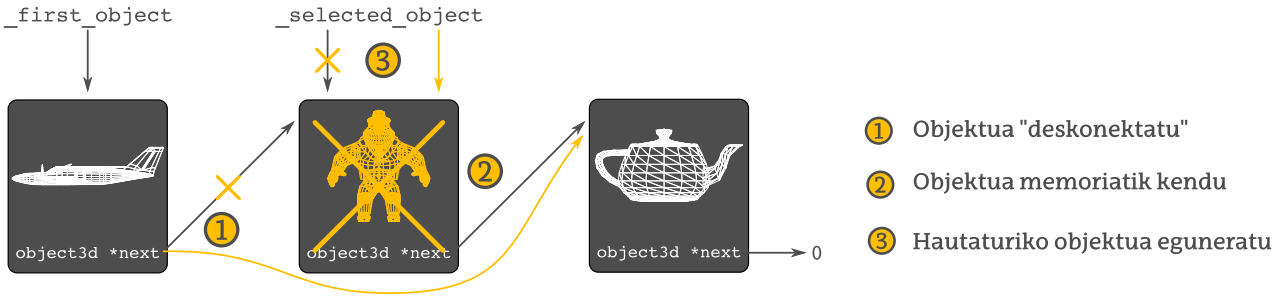
Fitxategiak irakurtzeko funtzioak zenbaki bat itzultzen du, prozesua nola joan den jakinarazteko; erroreren bat egon bada (hau da, funtzioak 1, 2 edo 3 itzultzen badu), mezu bat pantailaratuko dugu. Dena ondo joan bada (funtzioak 0 itzultzen badu), objektu berria gure zerrendan sartzen dugu eta mezu bat pantailaratzen dugu.

Objektua zerrendan sartzeko bi gauza egin behar dira. Lehenik eta behin, objektu berriari uneko zerrenda lotu, bere `next` eremuan uneko hasierako objektuaren pointerra gordez. Gero, objektu berria gure lehenengo objektutzat hartu behar dugu; `_first_object` pointerrean objektu berriaren helbidea gordeko dugu, alegia. Amaitzeko, objektu bat kargatzen dugunean hautaturikoa izatea nahi dugunez, `_selected_object` pointerra ere objektu berriari apuntatuko dio.

5.3.2 Objektuak aukeratzen eta ezabatzen

Kargaturik dauden objektuen artean bat hauta dezakegu tabuladore tekla erabiliz. Hona hemen kasu horretan exekutatzen den kodea:

```
case 9:
  _selected_object = _selected_object->next;
  if (_selected_object == 0) _selected_object = _first_object;
  break;
```



3 irudia: Objektu bat zerrendatik ezabatzeko prozesua

Kodea oso sinplea da: TAB tekla (9 kodea duena) sakatzen dugun bakoitzean `_selected_object` aldagaia apuntatzen dion objektuaren hurrengora apuntatuko dio (hots, bere `next` eremuan dagoenari). Hautaketa zirkularra izatea nahi dugu eta, hortaz, uneko objektuaren `next` eremuan 0 badugu (hau da, ez badago objektu gehiagorik), berriro lehenengo objektua hautatuko dugu.

Objektuen aukeraketak balio zaigu, besteak beste, objektuak ezabatzeko. Objektu bat zerrendatik kentzeko SUPR tekla sakatu behar dugu; ekintza horren eraginez, honako kode hau exekutatuko da:

```
case 127:
  if (_selected_object == _first_object)
  {
    _first_object = _first_object->next;
    free(_selected_object);
    _selected_object = _first_object;
  } else {
    auxiliar_object = _first_object;
    while (auxiliar_object->next != _selected_object)
      auxiliar_object = auxiliar_object->next;
    auxiliar_object->next = _selected_object->next;
    free(_selected_object);
    _selected_object = auxiliar_object;
  }
  break;
```

Objektuak ezabatzeko prozesua sinplea da. Lehenik eta behin, zerrendan hautaturik dagoen objektuaren aurrekoari antzeman behar diogu; gero, objektu honen `next` eremuan hautaturiko objektuaren pointerra gorde beharrean, beren hurrengoarena (aukeratuta dagoen objektuaren `next` eremuak apuntatzen diona) gorde beharko dugu. Honekin objektua listatik kendu dugu, baina ez memoriatik; memoria askatzeko `free` funtzioa erabili behar dugu. Amaitzeko, pointerrekin arazorik ez izateko `_selected_object` pointerra eguneratu behar dugu.

5.3.3 Bistaratze-eremuaren aldaketak

Teklatuaren funtzioekin amaitzeko bistaratze-eremua nola handitu ikusiko dugu. CTRL + - tekla konbinazioa sakatzen dugunean, ondoko kodea exekutatzen da:

```
case '-':
  if (glutGetModifiers() == GLUT_ACTIVE_CTRL){
    wd=(_ortho_x_max-_ortho_x_min)/KG_STEP_ZOOM;
    he=(_ortho_y_max-_ortho_y_min)/KG_STEP_ZOOM;

    midx = (_ortho_x_max+_ortho_x_min)/2;
    midy = (_ortho_y_max+_ortho_y_min)/2;

    _ortho_x_max = midx + wd/2;
    _ortho_x_min = midx - wd/2;
    _ortho_y_max = midy + he/2;
    _ortho_y_min = midy - he/2;
  }
  break;
```

CTRL + - konbinazioa sakatzen dugunean - karakterearen kodea jasotzen dugu. Hortaz, kodean lehendabiziko pausoa ea CTRL tekla sakaturik dagoenetz jakitea da. Horrela bada, `_ortho` aurrizkia duten aldagaiak eguneratuko ditugu, hauek definitzen baitute bistaratze-eremua[7].

Bistaratze-eremua aldatzean planoaren zentroa toki berdinean mantentzea nahi dugu, ikus puntua alda ez dadin. Hori lortzeko hiru pauso hartuko ditugu. Lehenik eta behin, planoaren zabalera eta altuera berriak kalkulatuko ditugu, unekoak erreferentziatzat hartuz (`wd` eta `he` aldagaiak, kodean). Gero, uneko planoaren erdiko puntuaren koordenatuak (`midx` eta `midy`) kalkulatu eta, amaitzeko, planoaren mugak eguneratu, erdiko puntuaren koordenatuei gehituz/kenduz altueraren edo zabaleraren erdia.

5.4 display.c

Fitxategi honek objektuak marrazteko behar diren funtzioak jasotzen ditu. Fase honetan hiru funtzio ditugu:

- `draw_axes` - Bere izenak dioen legez, funtzio honek ardatzak marrazten ditu
- `reshape` - Pantailaren tamaina aldatzean exekutatzen den *callback* funtzioa
- `display` - Pantaila marraztean exekutatzen den *callback* funtzioa

Azter ditzagun goiko funtzioak banan-banan.

5.4.1 Ardatzak marrazten

Ardatzak adierazteko marrak erabiliko ditugu. Hona hemen X ardatza marrazteko kodea:

```
glColor3f(KG_COL_X_AXIS_R,KG_COL_X_AXIS_G,KG_COL_X_AXIS_B);

glBegin(GL_LINES);
glVertex3d(KG_MAX_DOUBLE,0,0);
glVertex3d(-1*KG_MAX_DOUBLE,0,0);
glEnd();
```

Oso kodea sinplea izan arren, *OpenGL*n elementoak nola definitzen diren erakusten du. Lehendabiziko aginduak,`glColor3f`k, marrazteko erabiliko den kolorea ezartzeko erabiltzen da. C lengoaian, beste lengoaietan ez bezala, metodoak ezin dira gainkargatu (ezin dira definitu bi funtzio izen berdinarekin eta parametro ezberdinekin, alegia). Hori dela eta, *OpenGL*n definitutako metodo askotan, funtzioaren izenaren ostean, *kode* bat izango dugu, funtzioak zein parametro mota erabiltzen duen azaltzeko; kasu honetan `3f` daukagu, 3 `GLfloat` motako parametroak behar direla zehazteko. X ardatzaren kolorea, beste ezaugarri guztiek bezala, *definitions.h* fitxategian definiturik dago.

*OpenGL*n puntuen edo erpinen bidez definituriko objektuak sartzeko `glBegin` eta `glEnd` funtzioak erabiltzen dira. Lehenengoak parametro bat du, zein objektu mota marraztu behar den zehazten duena (marra bat kasu honetan). Bi aginduen artean elementoaren erpinak definitu behar dira, `glVertex3d` funtzioaren bidez[8]. Marra bat marrazteko bi puntu bakarrik behar ditugu; X ardatzaren kasuan, bi puntu horien Y eta Z koordenatuak 0 izango dira.

5.4.2 `reshape` funtzioa

Funtzio hau sinplea da oso:

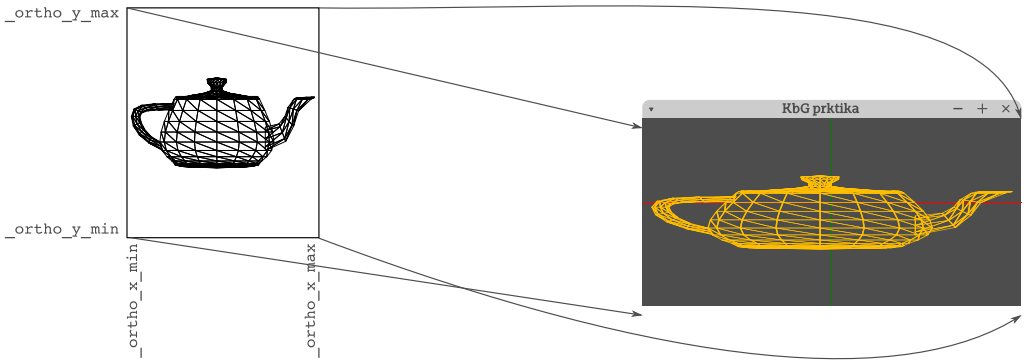
```
void reshape(int width, int height) {
    glViewport(0, 0, width, height);
    _window_ratio = (GLdouble) width / (GLdouble) height;
}
```

Funtzioak bi parametro ditu, leihoaren zabalera (`width`) eta altuera (`height`) berriak. Lehenengo komandoak objektuak marrazteko leihoaren zein zatia erabiliko dugun finkatzeko erabiltzen da; gure kasuan, leiho osoa erabiliko dugu[9]. Bigarren lerroan `_window_ratio` aldagai globala eguneratzen dugu, gero proiekzio-planoaren proportzioa egokitzeko.

[7] `_ortho_z_min` eta `_ortho_z_max` aldagaiek ez dute inongo eraginik proiekzio-planoan eta, hortaz, ez dira aldatu behar

[8] Kasu honetan parametroak `GLdouble` motakoak izango dira, `3d` kodeak adierazten duen bezala

[9] Pantaila beste eratan banatzerik ere badago. Esate baterako, beheko partean zati bat utzi dezakegu bertan mezuak pantaila-ratzeko, edo leihoa zatitan banatu proiekzio bat baino gehiago bistaratzeko



4 irudia: Proiekzio-planoak eta leihoak proportzio ezberdinak dituztenean objektuak marraztean deformatu egiten dira

5.4.3 `display` funtzioa

Hauxe da marrazteko erabiltzen den funtzioa; ikus dezagun prozesua pausoz pauso. Lehen urratsa pantaila garbitzea da, `glClear` aginduaren bidez. Ondoren, proiekzioa definitu behar dugu. *OpenGL*k proiekzioak kudeatzeko matrize-pila bat erabiltzen du eta, hortaz, proiekzioa definitzeko matrize-pila hori manipulatu behar dugu:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

if ((_ortho_x_max - _ortho_x_min) / (_ortho_y_max - _ortho_y_min) < _window_ratio) {
  GLdouble wd = (_ortho_y_max - _ortho_y_min) * _window_ratio;
  GLdouble midpt = (_ortho_x_min + _ortho_x_max) / 2;
  glOrtho(midpt - (wd / 2), midpt + (wd / 2), _ortho_y_min, _ortho_y_max, _ortho_z_min, _ortho_z_max);
} else {
  GLdouble he = (_ortho_x_max - _ortho_x_min) / _window_ratio;
  GLdouble midpt = (_ortho_y_min + _ortho_y_max) / 2;
  glOrtho(_ortho_x_min, _ortho_x_max, midpt - (he / 2), midpt + (he / 2), _ortho_z_min, _ortho_z_max);
}
```

Programa abiaraztean proiekzio matrize-pilan zaborra egon daitekeenez, lehendabiziko pausoa identitate-matrizearekin hasieratzea da. Hori lortzeko proiekzio matrize-pila modua aktibatzen dugu (kodearen lehenengo lerroa) eta bertan identitate-matrizea kargatzen dugu (bigarrengo lerroa).

Behin pila hasieratuta, bertan proiekzioa definitzen duen matrizea pilaratu behar dugu. Hori eskuz egitea egon arren, `glOrtho` funtzioaren bidez egingo dugu. Funtzio honek sei parametro ditu:

- `left` - X minimoaren balioa.
- `right` - X maximoaren balioa.
- `bottom` - Y minimoaren balioa.
- `top` - Y maximoaren balioa.
- `nearVal` - Hurbilen dagoen Z mugaren distantzia.
- `farVal` - Urrutien dagoen Z mugaren distantzia.

Lehenengo laurak proiekzio-planoa definitzen dute eta, gure programan, informazio hori `_ortho` aurrizki duten aldagaietan dago. Azken biak bistaratze-eremua begiradaren ardatzean[10] mugatzen dute. Muga horiek kamerarekiko ezartzen dira, gertuen eta urrutien dauden planoak kameratik zein distantziara dauden zehaztuz.

[10] *OpenGL*n *kamera*k -Z ardatzerantz begiratzen du

Proiekzio-planoak pantailaren proportzio berbera izan behar du, objektuak deforma ez daitezen (ikusi 4 irudia). Proportzioa berdina ez bada, beraz, zabalera edo altuera aldatu beharko dugu, leihoaren proportzioa duen proiekzio-plano berri bat definitzeko. Horixe da, hain zuzen, `if` egituran dagoen kodeak egiten duena.

Behin proiekzio matrize-pila ezarrita, objektuak marraz daitezke. Marrazten ditugun objektuei aldaketak eragin diezazkieke; aldaketa hauek ere matrizeen bidez definitzen dira. Oraingo honetan matrize-pila `MODELVIEW` deritzona da. Momentuz aldaketarik nahi ez ditugunez, edozer marraztu aurretik matrize-pila identitatearekin hasieratu behar dugu:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Gero, ardatzak marraztu ondoren, objektuen zerrenda korritu behar dugu, objektuak banan-banan sisteman sartzeko:

```
while (aux_obj != 0) {
  if (aux_obj == _selected_object){
    glColor3f(KG_COL_SELECTED_R,KG_COL_SELECTED_G,KG_COL_SELECTED_B);
  }else{
    glColor3f(KG_COL_NONSELECTED_R,KG_COL_NONSELECTED_G,KG_COL_NONSELECTED_B);
  }

  for (f = 0; f < aux_obj->num_faces; f++) {
    glBegin(GL_POLYGON);
    for (v = 0; v < aux_obj->face_table[f].num_vertices; v++) {
      v_index = aux_obj->face_table[f].vertex_table[v];
      glVertex3d(aux_obj->vertex_table[v_index].coord.x,
             aux_obj->vertex_table[v_index].coord.y,
                       aux_obj->vertex_table[v_index].coord.z);
      }
      glEnd();
    }
    aux_obj = aux_obj->next;
}
```

Objektu bakoitzeko, lehenik, zein kolorearekin margoztuko dugun ezarri behar dugu; hautaturikoa bada kolore batekin, eta beste batekin hautaturikoa ez bada. Gero, objektua sartu behar dugu, bere aurpegi guztiak korrituz. Aurpegi bakoitzeko poligono bat definituko dugu, zeinen erpinak aurpegiarenak izango diren. Gogoratu `face` egituraren `vertex_table` eremuan ditugun balioak ez direla erpinak, beraien indizeak baizik.

Behin gure eszena definiturik, funtzioan dugun azken komandoak, `glFlush`ek, objektu guztiak leihoan marrazten ditu.

6 Programa konpilatzen

Lehen aipatu bezala, gure programa konpilatzeko hiru liburutegi behar ditugu, `GL`, `GLU` eta `glut`. Liburutegiak, Ubuntu banaketan oinarrituriko linux sistematan, terminalean agindu hau exekutatuz instala daitezke:

```
sudo apt-get install mesa-common-dev libgl1-mesa-dev libglu1-mesa-dev freeglut3-dev
```

Programa konpilatzeko agindu hau exekutatu behar da, *.c eta *.h fitxategi guztiak dauden direktorioan:

```
gcc -o KbGprograma *.c -lGL -lGLU -lglut
```

Agindu honek `KbGprograma` izeneko exekutagarri bat sortuko du.

7 Kodeketa estiloa

Praktikaren hurrengo faseetan garatutako kodea entregatu behar da. Kodearen txukuntasuna eta estiloa mantentzearren, bete behar diren arau batzuk jasotzen dira hemen:

- Aldagai global guztien izenak (funtziotatik kanpo erazagututa daudenak, alegia) _ sinboloaz hasi behar dira, errazago antzemateko.

- Aldagaien, funtzioen eta konstanteen izenetan hitzak banatzeko _ karakterea erabili behar da.
- Konstante guztiak `definitions.h` fitxategian egon behar dira, motaren edo erabileraren arabera antolaturik.
- Definitzen diren konstante guziak letra larriz idatzita egon behar dira; beraien izenak `KG_` aurrizkiarekin hasi behar dira.
- Funtzioak ondo dokumentatuta egon behar dira. Funtzio guztien aurretik formatu hau duten iruzkinak idatzi beharko dira:

```
/**
 * @brief Funtzioaren deskripzioa
 * @param parametro_izena Parametroaren deskripzioa (parametro bakoitzeko lerro bat)
 * @return Funtzioak zer edo zer itzultzen badu, itzultzen duenaren azalpena
 */
```

- Goiko puntuan azaldutakoaz gain, funtzioen barruan aspektu nagusienak ondo dokumentatuta egon behar dira
- Debugeatzeko trazak erabiltzen badira (`printf`ak, alegia), entregatzen den bertsiotik kendu behar dira

8 Fase honen eginkizunak

Fase honetan puntu hauetan azaltzen diren atazak burutu behar dira:

- Pasatutako iturburu-kodea konpilatu eta exekutatu. Programaren erabilera probatu
- Alda itzazu atzeko planoaren, objektuen eta ardatzen koloreak; alda itzazu ere leihoaren izena eta dimentsioak.
- Programak akats bat du: objekturik ez badago eta SUPR tekla sakatzen badugu, programa ixten da. Topa eta konpon ezazu arazoa. Zer gertatzen da SUPR sakatu beharrean, TAB sakatzen bada?.
- CTRL + - sakatzean bistaratze eremua handitzen da baina, CTRL + + sakatzean, eremua ez da txikitzen. Inplementa ezazu funtzionalitate hori.