eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

PROGRAMAZIOAREN OINARRIAK: C-ko eta JAVA-ko praktikak

Nekane Bilbao, Cristina Perfecto, Gaizka Abaroa

EUSKARA ETA ELEANIZTASUNEKO ERREKTOREORDETZAREN SARE ARGITALPENA

Liburu honek UPV/EHUko Euskara eta Eleaniztasuneko Errektoreordetzaren dirulaguntza jaso du

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Programazioaren Oinarriak

C-ko Praktikak

Telekomunikazio Ingeniaritza Teknikoa - 1. maila

Bilbo, 2010eko iraila

Programazioaren Oinarriak: C-ko Praktikak

Laburpena

Praktika-bilduma honen helburua zera dugu: ikasleek *C-ko klase teorikoetan lantzen dituzten ezagutzak sendotzea*.

Bilduma honetan agertzen diren hamar praktikak bukatutakoan, ikasleak kontsolatik *Ontzidia Urperatu!* jokoan jostatzeko gutxigorabehera 400 bat kode-lerro dituen C programazio-lengoaian idatziriko aplikazioa izango du. Jokoak, besteak beste, erabiltzaile bat baino gehiagoren puntuazioaren jarraipena egitea, hasi eta bukatu gabeko partidak gorde eta gerora berreskuratzea, zein kanpo-erabiltzaile batek bere pantailak diseinatzea ahalbidetuko du.

Azken aplikazio horretara iristeko, ikasleak praktikaz praktika teoriako klaseetan landutako kontzeptuak erabiliko ditu, eta, programa erraz eta labur batetik abiaturik, azken kodea osatuko du. Proposaturiko hobekuntza berri bakoitzaren helburua da aplikazioaren aurreko bertsioaren aldean zer ekarpen egiten duen azpimarratzea.

GOMENDATURIKO JOKAERA: Praktika batetik hurrengo praktikara pasatu eta aurreko praktika horretan garatutako aplikazioa ez galtzeko, praktika berri bakoitzaren hasieran ikasleak aurreko praktikan erabili dituen *fitxategi guztien kopia* egin beharko du barnean duen direktorioan, kopia hori eta bertako fitxategiak funtzionaltasun berriak gauzatzeko erabiliko baititu.

Gaien Aurkibidea

0. praktika

Eclipserekin lanean hasteko oinarrizko gidaliburua

Programazioaren Oinarriak irakasgaiko C-ko zein Javako praktikak egiteko erabiliko dugun inguruneak *Eclipse* izena du. Eclipse IDE-a, jatorriz, Javan idatzitako aplikazioak garatzeko sortu zen, baina egun, osagarri edo *plugin* bidez, Java ez den beste programazio-lengoaia batzuetan garatutako programak edo aplikazioak sortzea ere posible da. Beraz, 0. praktika honetan, Eclipserekin lanean hasteko ezinbesteko pausoak laburbiltzen ahaleginduko gara, hasierako egunetan programazio-ingurunea erabiltzean sortzen diren zailtasunak ahal diren heinean leuntzeko.

Eclipseri buruzko informazio gehiago lortzeko zein Eclipse deskargatzeko, jo `http://www.eclipse.org/` webgunera.

0.1. Eclipse programa martxan jartzea

Eclipserekin lanean hasteko egin beharreko lehen gauza, noski, aplikazioa bera abian jartzea izango da. Horretarako, birritan klik egin beharko dugu Ubuntuko mahaigainean agertzen den zuzeneko atzipenerako ikonoan. Ia sakatzeaz batera, Eclipseren inguruko oinarrizko informazioa duen leihoa —esaterako, bertsioa (3.2, gure kasuan), copyright-a, etab.— agertuko zaigu. Hortik aurrera, benetan Eclipserekin lanean hasiko gara:

1. **Workspace Launcher** izeneko leihoa zabaltzen denean, Eclipsek *workspace* izenaz ezagutzen dena hautatzeko aukera ematen digu. *Workspace*-a, direktorio edo karpeta bat baino ez da, gure lanerako erabiliko dugun lehenetsitako karpeta, hain zuzen ere. Gure kasuan, praktika bakoitza workspace berri batean sartuko dugu. Erabilitako workspace-rako ibilbidea honelakoa izango da:

   `/home/user1/workspace/<ikaslearen izena>/<praktika zenbakia>`

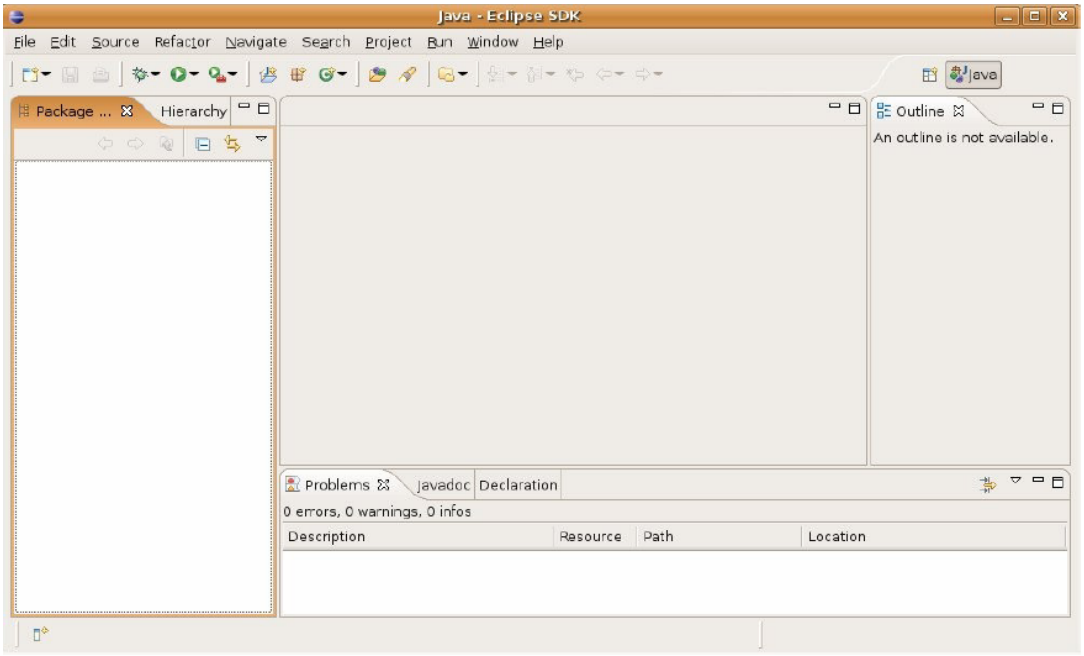
Esaterako, Iker izeneko ikasle batek 0. praktikan lan egin nahi badu, bere workspace-a finkatzeko, ibilbide hau idatzi beharko du:

```
/home/user1/workspace/Iker/0
```

Erabiliko dugun workspace-a sortuberria ez bada —noizbait aurreko praktika baterako erabili dugulako, alegia—, Eclipsek proposatzen digun ibilbidearen eskuinera dagoen gezian klik eginez, erabili diren azken workspace-n zerrenda agertuko zaigu. Zerrendan topatuko ez bagenu, '...' sakatu eta bilatzeko aukera izango genuke.

2. Workspace berri batera sartzen garen lehen aldian, Eclipsek ongietorri-orria, **Welcome** orria, erakutsiko digu. Gure praktiketan ez dugu hura erabiliko; beraz, aurrera jarraitzeko, itxi egingo dugu, leihoaren goiko eskuineko erpinean agertzen den X-n klik eginez.

3. Aurreko leihoa itxi, eta Eclipseren **Garapenerako Ikuspegia** izeneko leihoan izango gara. Bertan, informazioa zenbait eremutan kokatzen da, irudian ikus daitekeen modura.

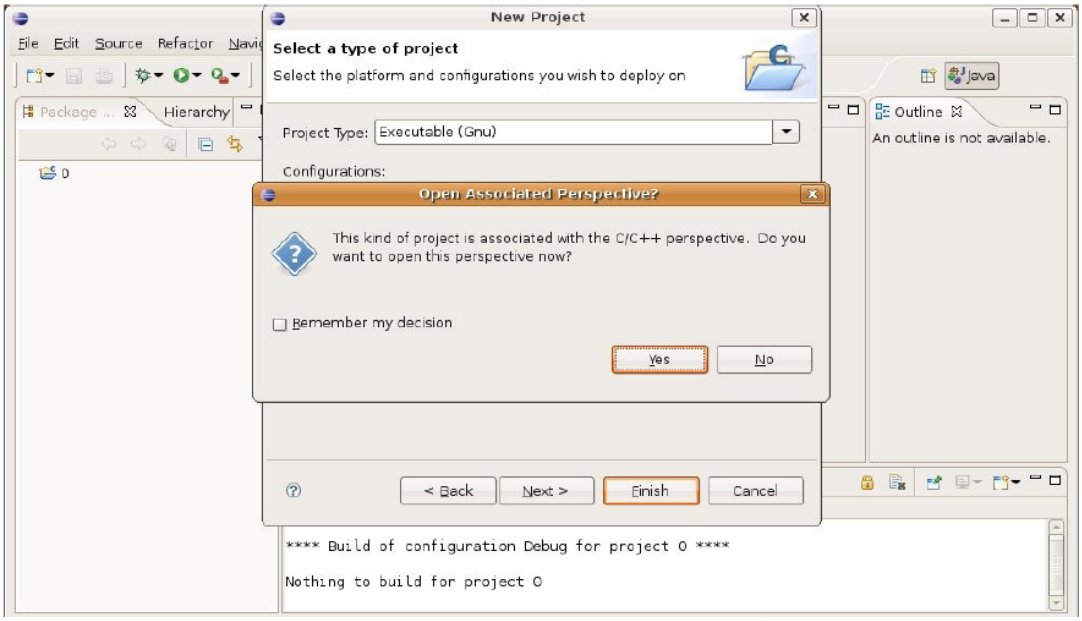


Irudia 1: Eclipseren Garapenerako Ikuspegia

0.2. Proiektu bat sortzea

Behin Eclipse programa erabiltzeko prest izanda, C-n programa bat idazten hasteko lehen egitekoa proiektu bat sortzea da. Horretarako, pauso hauek emango ditugu:

1. Hasteko, *File->New->Project* aukeratuko dugu.
2. **New Project** izeneko leihoa zabaldu, eta, bertan, *C-> Managed Make C Project* aukeratuko dugu haren gainean klik bikoitza eginez.
3. Segidan, proiektuari izen bat jartzea eskatuko zaigu. Workspace-aren ibilbidea aukeratzeko orduan ezarri den irizpidea, praktikaren zenbaki bera erabiltzea izan da; proiektuaren izena jartzeko irizpidea, berriz, praktika barneko ariketaren zenbaki bera erabiltzea izango da. Horrela, ariketa berri bakoitzeko proiektu ezberdin bat sortu beharko dugu, eta hari ariketaren zenbakia jarriko zaio izen modura. Izena jarri, eta **Next** sakatuko dugu aurrera jarraitzeko.
4. Agertuko zaigun hurrengo leihoa, inolako aldaketarik egin gabe, **Finish** sakatuta itxiko dugu.
5. Eclipsek **Open Associated Perspective** izeneko leihoa zabaltzen badu, **Remember my decission** markatu eta baiezko erantzuna hautatuko dugu. Hala, workspace bereko proiektu bakoitzerako berriz erantzun beharra saihestuko dugu. Ikus irudia:



Irudia 2: Eclipseren Open Associated Perspective

6. Proiektua sortzeaz batera, pantailaren ezkerreko zutabean karpeta-itxurako ikono bat agertuko zaigu ireki berri dugun proiektuaren izenaz. Gure lehen proiektua sortu dugu!

0.3. Iturri-fitxategi bat sortzea

Proiektua sortua izanik, iturri-fitxategiaren idazkerari ekiteko ordua iritsi da.

1. Proiektuaren izenaren gainean jarri, arratoiaren eskuineko botoian klik egin, eta agertuko zaigun menuan *New->Source File* aukeratu beharko dugu.

2. Beste leiho bat irekiko zaigu, eta iturri-fitxategiaren izena zehazteko eskatuko digu. Fitxategiari, proiektuak duen izen bera emango diogu, izenaren bukaeran **".c"** luzapena erantsiz. Segidan, **Finish** sakatuko dugu.

3. Pantailaren ezkerreko zutabean, gure proiektu barruan iturri-koderako fitxategia sortu dela baina oraindik hutsik dagoela adierazten duen ikonoaz batera, karpeta berri bi agertuko dira. Pantailaren panel nagusia berriz, zuriz agertzen da. Gure programa bertan idazten hasteko prest dago.

   Programa sinple batetik abiatuko gara:

   ../src/p0/main.c

```
#include <stdio_ext.h>
int main(){
    char izena[20];
    int adina;
    printf("Idatzi zure izena: ");
    scanf("%s",izena);
    printf("Idatzi zure adina: ");
    scanf("%d",&adina);
    printf("Kaixo %s! %d urte dituzu!!!!\n",izena,adina);
    return 0;
}
```

4. Fitxategian egindako aldaketak gordeko ditugu; horretarako, disko-itxura duen ikonoa erabiliko dugu. Eclipsek, kontrakoa adierazten ez badiogu, iturri-fitxategi guztiak birkonpilatzen ditu haietako baten bat aldatu eta gordetzeko agindua jaso eta gero. Horregatik, oso gomendagarria da proiektu bakarra zabalik izatea.

5. Behin konpilatuta, iturri-fitxategiaren beheko aldean kokaturik dagoen **Problems** leihoan, konpilazio- eta estekatze- prozesuetan sorturiko arazoak bistaratuko dira. Arazo horiek era bitan adieraziko zaizkigu; akats eta *warning* edo abisu gisa. Gure programa idaztean akats sintaktikoak gauzatu baditugu, konpiladoreak akats gisa adieraziko dizkigu, eta ez du exekutagarririk sortuko, hau da, oraindik ez dugu exekutatzeko programarik izango. *Warning*-ak arazo ez hain larrien adierazgarri dira; konpiladoreak izaniko arazoek *warning*-ak bakarrik sortu badituzte, exekutagarri bat sortua izango da, eta gure programa exekutatu ahalko dugu. Dena den, *warning* gabeko konpilazioa gomendagarria da, *warning*-ik balego, programaren funtzionamendua ez bailitzateke zuzena izango.

0.4. Akatsik ez duen iturri-fitxategi bat konpilatzea eta exekutatzea

1. Aurreko kodea zuzen kopiatu badugu, **Problems** leihoan *warning* bakarra izan dezakegu, fitxategia lerro huts batean bukatzen ez dela adieraziko diguna... Beraz, exekuta dezakegun programa bat izango dugu!

2. Menu honetara joan: *Run->Run...*, eta **Run** leihoa irekiko zaigu. Ezkerreko zutabeko **C/C++ Local Application** lerroaren gainean birritan klik egin beharko dugu, eta, behealdean, proiektuaren izena izango duen lerro bat agertuko zaigu. Eskuinaldean, 'Main' leiho bat agertuko da, zeinean **C/C++ Application** balioa sartu beharko dugun. Horretarako, **Browse** botoia sakatuko dugu, eta irekiko zaigun leihoan, proiektuaren izen bera izango duen bitarra —**Binary** aukeraren bidez— hautatuko dugu. Azkenik, **Run** botoia sakatuko dugu gure programa exekutatzen has dadin.

3. Exekuzioa **Console** leihoan gauzatuko da, **Problem** deritzonaren ondoan. Ikusteko, izena sakatu baino ez dugu egin beharko...

0.5. Akats sintaktikoak dituen iturri-fitxategi bat konpilatzea

1. Akats sintaktiko bat behartuko dugu; horretarako, *adina* aldagaiaren deklarazio-lerroa kenduko dugu, adibidez. **Problems**-en akats bat agertuko da gorde eta konpilatzean.

2. Akatsa azalpen labur batez eta gauzatu den lerro-zenbakiaz agertuko da. Akatsaren gainean birritan klik eginez gero, akatsa aurkitzen den lerrora eramango gaitu. Hala ere, errorea aurkitu den tokiak ez du beti adierazten konpontzeko egin beharreko aldaketen puntua...

0.6. Edizio-akatsak dituen iturri-fitxategi bat konpilatu, exekutatu eta araztea

1. Akats sintaktiko bat eragingo ez duen idazketa-akats bat sortuko dugu. Akats horrek programaren funtzionamendu zuzena eragotziko du. Horretarako, azken *printf*-a beste honengatik aldatuko dugu:

```
printf("Kaixo %d, %d urte dituzu!!!!\n",izena,adina);
```

2. Gorde eta konpilatuko dugu. Oraingo honetan, **Problems**-en ez da akatsik egongo. Lorturiko programa lehen bezala exekutatuko dugu... eta dena behar bezala exekutatuko da azkenera iritsi arte. Oraingoan, agurrean ez da sarturiko izena agertuko, zenbakia baizik. Baina guk izen bat sartu diogu; zer gertatu da orduan?

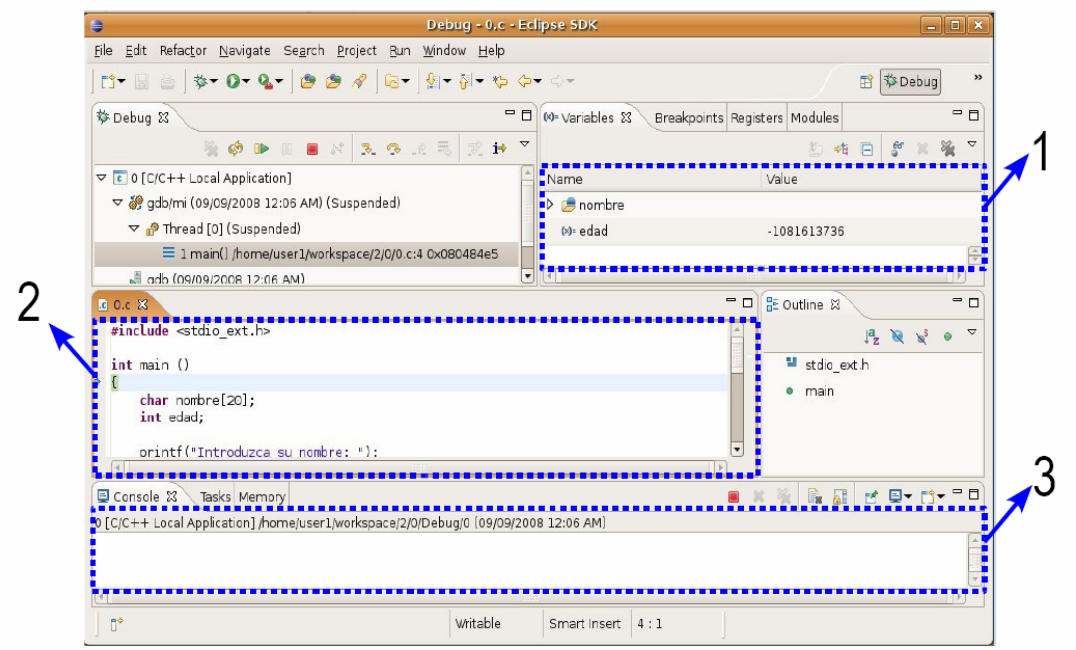
3. Arazketa bidez konpontzen den funtzionamendu-akats baten kasuan gaude. Kasu honetan, izena gaizki gorde eta gaizki bistaratzen delako gerta daiteke arazoa. Arazteko, oraingoan aukera honetan sartuko gara: *Run->Debug* eta **Debug** leihoa irekiko zaigu, —**Run**-en oso antzekoa dena—. **Debug**-en, **Run**-en exekutatzeko egiten dugun gauza bera egingo dugu. Gero, arazketa hasteko, **Debug** botoia sakatuko dugu.

4. Workspace batean arazketa-prozesua martxan jartzen den lehen aldian, Eclipsek ikuspegia aldatzearekin ados gauden galdetuko digu... 0.2. ataleko 5. puntuan bezala. Arazketa-ikuspegiak zenbait eremu ditu; irudi honetan ikus daitezke:

   1: Memorian kokaturiko aldagaiak

   2: Iturri-fitxategia

   3: Kontsola



Irudia 3: Eclipseren Arazketa Ikuspegiaren Eremuak

5. Kurtsorea iturri-fitxategiaren lehen '{'-ren gainean egongo da. **F6** sakatuz, programa pausoz pauso exekutatuko da. Arazteko, garrantzitsua da, exekutatzen den pauso bakoitzeko (F6 sakatuz), kontsolan dagoena eta aldagaien balioak baieztatzea. Horrela, baiezta dezakegu *scanf*-ek teklatutik zerbait sartu arte itxaroten duela, adibidez. Gure kasuan, aldagaietan gorderiko balioak teklatutik sarturikoekin bat datozela egiaztatu ahalko dugu (aldagaien gainean sakatuz).

6. Azken *printf*-era iristean, ikusiko dugu *izena* aldagaiak izen zuzena gordea duela; hala, arazoa *printf*-ean dagoela ikusiko dugu. Hau da, *s*-ren ordez *d* jartzeak eragin du funtzionamendua ez zuzena izatea, eta arazketari esker jakin dugu.

Orain, Eclipseren oinarrizko erabilera menperatzen duzula, prest zaude zeure programak idazteko!

1. praktika

Oinarrizko esleipen eta eragiketak

Lehen praktika honetan, ikaslea aldagai sinpleak sortzen eta erabiltzen trebatu nahi dugu. Gainera, oso erabilgarriak suertatuko zaizkion eta laster funtzioen gaian sakonago landuko dituen funtzio batzuk ere —teklatutik irakurtzeko (sarrerako) eta pantailan idatzi/bistaratzeko (irteerako) funtzio batzuk— erabiltzen hasiko da.

1.1. Aldagaien deklarazioa

Hasteko, *Ontzidia Urperatu!* jokoan gure itsasontziak kokatzeko erabiliko dugun oholaren tamaina definituko dugu. Irudi honek gure oholaren itxuraren lehen zirriborroa erakusten digu:

	0	1	2	3	4	5	6	7	8	9
A										
B										
C										
D										
E										
F										
G										
H										
I										
J										

Main funtzio nagusian, sortu aldagai hauek:

- `int` motako bat, oholaren azken zutabearen zenbakizko balioa gordeko duena. Gure ohola 20 zutabekoa bada, eta zutabe horiek 0tik zenbatzen hasten direla kontuan hartuz (0, 1, 2, ..., 19), azkenengoa 19.a izango litzateke.

- `char` motako bigarren bat, oholaren azken errenkadari dagokion letra gordetzeko erabiliko dena. Ohola 10 errenkadakoa bada, (A, B, C, ..., J) azkenengoa J.a izango da.

Iturri-kodean bertan has zaitezke aldagai horiei hasierako balio bat ematen —horri "hasieratu" deituko diogu hemendik aurrera—. Eragiketa hori, aldagaiaren deklarazioaren beraren bitartez edo esleitze-eragiketa berri eta bereizi baten bidez egiteko aukera izango dugu.

Saiatu programa konpilatzen, den-dena akatsik eta *warning*-ik gabe utzi arte. Egiaztatu funtzionamendua ere zuzena dela.

1.2. Pantailan bistaratzea

Oholaren dimentsioak finkatzeko erabili ditugun hasierako balioak aldagaiei esleitutakoan, erabiltzaileari zuzendutako mezua bistaratzen da. Esaterako:

```
Ongi etorri "Ontziteria Urperatu!" jokora

- Oholaren tamaina: 20 zutabe x 10 errenkada da.
- Erabili zenbakizkoa zutabea adierazteko (0, 1, 2, ..., 19)
- Erabili letra larria errenkada adierazteko (A, B, C, ..., J)
```

Aurreko mezuan adibide modura agertzen diren zenbaki eta letrek programako aldagaietan gorderiko balioen araberakoak izan behar dute. Balio horiek aldatu, eta birkonpilatu eta exekutatu programa, erakutsiko den mezua aldatuz joango dela egiaztatzeko.

1.3. Teklatutik irakurtzea

Orain, erabiltzaileari teklatuaren bidez koordenatu bat sartzeko aukera emango diogu; ostean, pantailan aldagaiari emandako balioa bistaratuko dugu.

Hori gauzatzeko, koordenatuaren balioak gordeko dituzten aldagai berri biak sortu beharko ditugu:

- `char` motakoa, errenkadaren balioa gordetzeko.
- `int` motakoa, zutabearen balioa gordetzeko.

Segidan, erakuts ezazu erabiltzaileari koordenatu bat sartzeko eskaera luzatzen dion testu-mezua. Adibidez:

```
Sartu, mesedez, koordenatu bat (adib. B12):
```

Irakurri erabiltzaileak sartu duen balioa. Horretarako, erabili `char` motako aldagaia letrarako eta `int` motako aldagaia zenbakirako.

Bukatzeko, aurrekoan erabili duzun antzerako mezu baten bidez, erakutsi koordenatua pantailan. Adibidez:

```
(B Errenkada, 12 Zutabea)
```

2. praktika

Baldintzazko sententziak

Baldintzazko sententzien bidez, konparaketak egin, eta, haien emaitzaren arabera, programak gauza bat edo beste bat egitea lor dezakegu. Praktika honetan, erabiltzaileak sartzen dituen datuak egiaztatzeko tresna gisa erabiliko ditugu.

2.1. Erabiltzailearen sarrerako datuak prozesatzea

Aurreko praktikaren kodea abiapuntutzat hartuz, erabiltzaileari letra maiuskulaz zein minuskulaz sartzeko aukera emango diogu. Horretarako:

- Lehenik, erabiltzailearen sarrera irakurri eta letra (errenkadari dagokiona) 'a' eta 'z' balioen artean dagoen begiratu beharko da. Hala bada, ASCII kodeari 32 kenduz, letra maiuskula bihurtuko dugu.
- Balio berri hori erabiliko da aurrerantzean, bai baliozko heinean dagoen egiaztatzeko (hurrengo atala) zein koordenatua pantailan bistaratzeko.

2.2. Erabiltzailearen sarrera irakurtzea

Sartutako koordenatuak baliozkoak diren ala ez egiaztatzeko behar den kodea gehitu. Baliozkoak ez badira, horretaz ohartu eta erabiltzaileari jakinarazi beharko zaio. Konkretuki:

- Errenkada A-ren eta azken errenkadari dagokion letraren artean dagoela egiaztatu beharko da. Baliozko heinetik kanpo baldin badago, errenkadetarako onartutako heina adieraziko duen mezua bistaratuko da.

- Zutabea 0ren eta azken zutabeari dagokion zenbakiaren artean dagoela egiaztatu beharko da. Horrela ez bada, zutabeetarako onartutako heina adieraziko duen mezua bistaratuko da.

2.3. Jokoa bukatzea

Bukatzeko, programa bukatu nahi dugun edo ez adierazteko, azken egiaztatze bat egingo dugu (hori aurrerago erabiliko da).

- Erabiltzaileak 'Z0' sartzen badu, mezu bat bistaratu eta programa buka dadin behar diren aginduak gehitu beharko dira.

3. praktika

Egitura errepikakorrak

Egitura errepikakorren bidez, kode-ale bat X aldiz errepikatu dezakegu; birritan, hirutan...; errepikapen bakoitzean parte hartzen duten aldagaien balioak exekuzioaren arabera aldatuz joango dira. Praktika honetan, lehenik ohola marrazteko eta ondoren erabiltzaileak koordenatu anitz sar ditzan erabiliko ditugu egitura horiek.

3.1. Ohola marraztea

Berriz, aurreko praktikako kodea hartzen dugu abiapuntutzat. Errenkadaren eta zutabearen adierazpen-formatua eta baliozko heina azaltzen dituen hasierako mezuaren ordez, oholaren marrazki bat jarriko dugu (azken finean, zer da irudi bat baino adierazgarriagorik?).

Lehen proba gisa, koordenatu-ardatzak marraztuko ditugu:

- `for` motako begizta baten bidez, zutabe bakoitzaren zenbakia bistaratuko da, formatu honi jarraituz:

  ```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
  ```

  Kontuan izan lerroaren hasieran hutsune bi daudela eta zenbakien artean hutsune bakarra. Horrez gain, kontuan hartu, halaber, digitu bakarreko zenbakiek hutsune bat gehiago eramango dutela. Hala, digitu bakarreko zenbakiek bikoek beste toki erabiliko dute.

- Aurreko begiztatik kanpo jarriko den beste `for` begizta baten bidez, bertikalki erakutsi errenkada bakoitzari dagokion hizkia. Horretarako, hizki bakoitzaren aurretik lerro huts bat utzi:

```
    A

    B

    C
```

Hauxe da azken emaitzaren adibidea:

```
Ongi etorri "Ontziteria Urperatu!" jokora

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19

A

B

C

D

E

F

G

H

I

J

Sartu, mesedez, koordenatu bat (adib. B12):
```

Ohola jadanik funtzionala da, baina hobe gelaxkak marrazten baditugu. Horretarako, bigarren `for` begizta aldatuko dugu, eta haren barnean beste begizta batzuk mailakatuko ditugu.

- Hutsik dagoen lerroa, hutsune batez hasi eta oholak dituen zutabe beste "|__" karakterez (komatxorik gabe) osaturik egongo den lerro batez ordezkatu. Adibidez:

  ```
  |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
  ```

- Hizkia adierazten duen lerroan, hizkiaren ostean, oholak dituen zutabe beste "| " karaktere (komatxorik gabe) itsatsi.

Berriz ere, azken emaitzaren itxura honako hau izango da:

```
Ongi etorri "Ontziteria Urperatu!" jokora

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
A|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
D|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
E|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Sartu, mesedez, koordenatu bat (adib. B12):
```

Errenkada eta zutabe kopurua aldatu eta egiaztatu horrek begiztaren iterazio kopurua aldatzen duela, eta beraz, baita marrazturiko oholaren tamaina ere.

3.2. Sarrera aniztunak onartzea

Behin egitura errepikakorrak ezagututa, erabiltzaileak koordenatu bat baino gehiago sar ditzan erabiliko ditugu.

- `char` motako `irten` deritzon aldagai bat sortuko da, eta 0 balioaz hasieratuko da.
- Erabiltzailearekin elkarrekintza gauzatzeko beharrezkoa den kodea `do-while` begizta baten barnean sartu; hala, koordenatuak sartzeko eskatzen joango zaio.
- Begizta amaitutzat joko da erabiltzaileak Z0 konbinazioa sartzen duenean. Une horretan erabiltzaileari mezua erakutsiko zaio, eta `irten` aldagaiak 1 balioa hartuko du.

Eginiko aldaketa dela-eta programak sortuko duen irteeraren adibidea:

```
Ongi etorri "Ontziteria Urperatu!" jokora

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
A|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
D|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
E|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Sartu, mesedez, koordenatu bat (adib. B12): g6

(G Errenkada, 6 Zutabea)

Sartu, mesedez, koordenatu bat (adib. B12): l8

(L Errenkada, 8 Zutabea)
Errenkadetarako baliozko heina hau da: A - J

Sartu, mesedez, koordenatu bat (adib. B12): l99

(L Errenkada, 99 Zutabea)
Errenkadetarako baliozko heina hau da: A - J
Zutabetarako baliozko heina hau da: 0 - 19

Sartu, mesedez, koordenatu bat (adib. B12): Z0

--- Aplikazioa bukatua erabiltzailearen aginduz ---
```

4. praktika

Funtzioak

Azkenik, funtzioak erabiltzeko aukera daukagu!

Funtzio nagusia; `main` deritzona, gehiegi hazten ari da, eta kodea zailtzen ari da. Oso arrunta da programa batek milaka lerro behar izatea, eta zaila da dena funtzio bakar batean sartzea. *Zatitu eta irabaziko duzu!* lema programazio egituratuaren oinarria da, eta funtzioak arazoaren zatiketa horren ondorio dira.

4.1. Diseinua berregitea

Programaren lehen lerroa idatzi aurretik, horren **diseinu** egoki bat egin behar da. Lan hori kodifikazioa bezain garrantzitsua da. Programa nola egituratu pentsatu behar da, hau da, zein zatitan banatuko den eta nola komunikatuko diren zati horiek. Orain, funtzioak egiten ikasi dugun honetan, programaren kodea berregituratuko dugu berandu izan aurretik.

Arau orokor gisa, `main` funtzioak ez luke pantaila bat baino gehiago bete beharko eta programak zer egiten duen erakutsi beharko luke. Funtzio nagusi horrek beste funtzio txikiagoen deiak izango ditu, eta funtzio txiki horiek *benetako* lana egingo dute.

Hori kontuan izanik, marrazki baten bidez orain arte egindako kodea zer funtziotan zatituko duzun adierazi behar duzu. Fluxugrama baten bidez, adierazi funtzio horiei nola deitzen zaien `main` programatik. Hori eginda, klasean proposaturiko egiturarekin konparatu eta bakoitzaren abantaila eta desabantailei buruz eztabaidatu.

4.2. Kodea berregituratzea

Behin diseinua bukatuta dagoenean, pentsatu dugun funtzioetan zatituko dugu kodea:

- `OholaMarraztu` funtzioak:
  - Sarrerako parametro bi hartzen ditu; bat, `char` motakoa, azken errenkadaren hizkiaren balioarekin; bestea, `int` motakoa, azken zutabearen zenbakizko balioarekin.
  - Ez du irteerako baliorik ematen.
  - Parametroen bidez adierazitako tamainako ohola marrazten du.
- `KoordenatuakIrakurri` funtzioak:
  - Oholaren azken errenkada eta zutabea hartzen ditu parametro moduan.
  - Irteerako balio bi itzultzen ditu; bat, `char` motakoa, aukeratutako errenkadaren balioarekin; bestea, `int` motakoa, aukeratutako zutabearen balioarekin.
  - Gainera, bere identifikatzaileari lotua, erabiltzaileak bukatu nahi duen ala ez adierazteko `char` motako beste balio bat itzuliko du: '0' balioduna irten nahi badu, eta '1' balioduna jarraitu nahi badu.
  - Teklatuaren bidez sartzen diren koordenatuak jaso eta egokiak diren baieztatuko du. Horrela ez bada, berriz eskatuko dizkio erabiltzaileari. Horretarako, aurreko praktikan idatzitako kodea erabiliko dugu.
- `main` funtzioa:
  - Berridatz ezazu funtzio bi horiek erabil ditzan. Ohola bistaratu eta koordenatuak eskatuko ditu erabiltzaileak bukatu nahi duela adierazi arte (horretarako, `KoordenatuakIrakurri` funtzioaren itzulera balioa erabiliko du).
  - Gogoratu funtzio honetan argi geratu behar duela programak zer egiten duen; era irakurterrazean eta sinplifikatuan.

4.3. Lehen itsasontzia sortzea

Orain arte, jokoak ez du erabilera handirik izan, erabiltzaileak sartzen dituen koordenatuekin ez du ezer egiten eta. Hori konpontzeko, lehen itsasontzi bat sortuko dugu, oso sinplea. Hurrengo praktiketan itsasontzi kopurua eta motak gehituko ditugu. Gure lehen itsasontziak gelaxka bakar bat beteko du. Horretarako, funtzio berri bat gehitu eta `main` funtzioa moldatuko dugu:

- `TiroaEgiaztatu` funtzioa:
  - Funtzio hori hain sinplea izanik, hark egiten duena `main` funtzioan zuzenean egiteko aukera legoke, baina hemen diseinua sartzen da jokoan berriro ere. Funtzio hori, aurrerago moldatu egingo da; horrela, bidea eginda uzten dugu.
  - Funtzioak sarrerako lau parametro hartzen ditu: erabiltzaileak sarturiko koordenatuak eta itsasontziaren posizioa duten koordenatuak.
  - Tiroaren puntuazioa itzultzen du zenbaki oso baten bidez: 0 balioa, itsasontzia jo ez badu, eta 1 balioa, jo badu.
- `main` funtzioak:
  - Itsasontziaren koordenatuak gordetzeko aldagaiak sortuko eta hasieratuko ditu.
  - *Biziak* aldagaia sortuko du; aldagaia `int` motakoa da, jokalari bati uzten zaizkion tiroekin hasieratua.
  - *Puntuazioa* aldagaia sortuko du; aldagai hori `int` motakoa da, eta 0 balioarekin hasieratua egongo da. Jokalariak lortzen duen puntuazioa gordeko du.
  - Begizta nagusian, eta behin erabiltzailearen koordenatuak irakurri eta gero, deitu `TiroaEgiaztatu` funtzioari. Hark itzultzen duen balioa egiaztatu beharko da, hau kontuan izanik:
    - 0 balioa bada, bizi bat kendu beharko zaio erabiltzaileari. Ez duela asmatu adierazi beharko zaio eta geratzen zaizkion biziak bistaratu. Gainera, erabiltzaileak bizi guztiak galdu baditu, programak bukatu beharko du.
    - 0 balioa ez bada, bere balioa puntuazioari gehituko zaio. Erabiltzaileari mezu bat bistaratuko zaio, asmatu duela jakinarazteko eta batutako puntuak adierazteko.
  - Bukatzeko, programa amaitu aurretik zergatik irteten den adierazi beharko da: erabiltzaileak eskatu duelako (*Z0*) edo biziak agortu zaizkiolako (*Game Over*). Azken horretan, lortutako puntuazioa ere bistaratu beharko da.

5. praktika

Array dimentsiobakarrak

Praktika honetan beste kontzeptu garrantzitsu bat aurkeztuko dugu; arrayena hain zuzen ere. Array batek mota berdineko aldagai sorta bat izatea ahalbidetzen du.

Haren erabilgarritasunaz ohartzeko, jokoak aldaketa bat izango du. Aldaketa hori izango da jokalariak tiro egiten duen unea gordetzea; horrela, jokoa bukatzean, jokalariak non tiro egin pentsatzeko behar izan duen denbora maximoa —tirotik tirora igaro den denbora maximoa— adieraziko da.

Horren ondoren, jokalariak egin ditzakeen tiro kopuruari muga jarriko zaio; balio hori preprozesadore konstante baten bidez definituko da.

5.1. Denborak gordetzeko array dimentsiobakarra

5.1.1. Sarrera: denbora lortzea

Tiroa zer unetan egin den jakiteko, ordenagailuen barne-erlojua erabiliko da; hain zuzen ere, ordenagailu baten erabiltzaileari ordua erakusteko edo fitxategi bat diskoan zer ordutan gorde den adierazteko erabiltzen den erloju berbera.

C-k, erloju horretako informazioa lortzeko, `time` deritzon funtzio bat definiturik dauka. Funtzio estandar bat da; hau da, C-ko liburutegietan dago jadanik —*printf*-ekin gertatzen den moduan—; beraz, ez dago haren kodea programatu beharrik.

Funtzio horri deitzen zaion bakoitzean, 1970eko urtarrilaren 1eko 0:00 ordutik igarotako segundo kopurua jasotzen da. `time` funtzio horri dagokion formatua, time.h

goiburuko fitxategian definiturik dago, itxura honen arabera:

```
time_t time(time_t *t);
```

Azalpen modura:

- `time_t` datu mota bat da (`int`-en edo `char`-en baliokidea), praktikoki, `long int` bat balitz bezala lan egin daiteke berarekin —8byte-ko tamaina du—. Hala, *printf* bat erabiliz bistaratzeko, erabiliko den formatua `%li` izango da.
- `*t` `time_t` motako aldagai bati zuzenduriko erakuslea. Praktikan, `NULL` parametroa erabiliz ere dei dakioke.

  Horren guztiaren adibide gisa, kode hau bistaratzen da:

  ../src/p5/main.c

```
#include <stdio.h>
#include <time.h>
int main(void){
    time_t segundoak;
    segundoak = time(NULL);
    printf(" 1970eko urtarrilaren 1eko 0:00 ordutik igarotako segundo kopurua:");
    printf("%li", segundoak);
}
```

Informazio gehiago behar izanez gero:

- `time` funtzioa: GNU/Linux-en, funtzio horri dagokion laguntza-dokumentazioa kontsola bidez ikus daiteke, horretarako, agindu hau erabili beharko da:

  `man 2 time <ENTER>`

- `time_t` mota: ikus ezazu `http://en.wikipedia.org/wiki/Time_t`

5.1.2. Ariketa

`main` funtzioak aldaketa hauek izango ditu:

- Tiro bakoitzari dagokion denbora gordetzeko, `time_t` motako elementuak gordeko dituen array dimentsiobakar bat adieraziko da. Arraya, gehienez, 20 tiro gordetzeko gai izango da; horretarako behar duen muga-balioak preprozesadore konstante baten bidez definiturik egon beharko du.

- Begiztaren barruan `time` funtzioak itzultzen duen balioa arrayaren posizio bakoitzean gordeko da; erabiltzaileak sarturiko koordenatuak irakurri orduko deituko zaio horri. Begiztaren funtzionamendua zuzena izan dadin, erabiltzen den saiakera kopurua gordeko duen eta 0ra hasieratua egongo den aldagai bat definitu beharko da. Aldagai horren balioa onartuta dagoen tiro kopuru maximora iritsi den balioztatu beharko da begiztaren barruan.

- Programa bukatu aurretik, `InfoMaximoa` funtzioari deituko zaio.

`InfoMaximoa` izendun funtzio berri bat idatzi beharko da; funtzio horrek honako hau egin beharko du:

- Denborak gordetzen dituen arraya eta jokalariak erabili dituen saiakeren kopurua jasoko ditu sarrerako parametro gisa.

- Ez du ezer itzuliko (`void`).

- Jokalariak erabili duen denbora maximoa —segundotan— erakutsiko da. Hori egiteko, arraya miatu, eta bertan jarraian gorderik dauden denboren arteko diferentzia kalkulatuko da. Ondoren, jarraian dauden elementu biren arteko diferentzia aurrekoarena baino handiagoa den balioztatu beharko da.

5.2. main funtziorako parametro-arraya

5.2.1. Sarrera: main funtzioaren sintaxia

C-ko `main` funtzioak bere sarrerako parametroak hainbat sintaxi erabiliz jaso ditzake; adibidez:

../src/p5/main-sintaxia.c

```
int main (void)
int main (int argc, char *argv[])
int main (int argc, char *argv[], char *envp[])
```

Horietatik guztietatik, ariketa hau egiteko bigarrena erabiliko dugu. Hau da sintaxiaren azalpena:

- `argc`: funtzioari deitzen zaionean jasotzen duen parametro kopurua gehi bat.

- `argv`: karaktere-katez (string-ez) osaturiko array bat. Haren osagaiak izango dira programaren izena eta deitzean erabiltzen diren parametroak. Hori dela eta, haren tamaina `argc` da.

Nola deitzen zaio programa bati eta nola pasatzen zaizkio parametroak? Leihoz osaturiko ingurune grafiko batean gaudenean, klik bat —GNU/Linux-en kasuan— edo klik bikoitza —Windows-en kasuan— eginez, programa bat exekutatzen ari gara, eta programa hori C-n eginda badago, programaren `main` funtzioari dei egiten zaio. Egia esanda, era horretara eginda ez da argi ikusten parametroak nola pasatzen zaizkion; hori ikusteko, kontsola erabili beharko da. Beraz, pauso hauek egingo ditugu:

1. Editatu eta gorde karpeta batean `main_args.c` izena duen eta lerro hauek izango dituen fitxategi bat:

   ../src/p5/main-args.c

   ```
   #include <stdio.h>
   int main (int n, char *args[])
   {
       int i;
       printf ("Argumentu Kopurua: %d\n",n-1);
       printf ("Programaren Izena: %s\n", args[0]);
       for (i=1; i<n; i++)
           printf ("Argumentua %d: %s\n", i, args[i]);
       return 0;
   }
   ```

2. Kontsola ireki, eta sarturiko kodea duen karpetara joan. Hori egin eta gero, programa agindu hau erabiliz konpilatuko da:

   ```
   gcc main_args.c -o main_args_exekutagarria
   ```

3. Kontsolan jarraituz, programari hainbat parametro erabiliz deituko diogu —horrela, kasu bakoitzean haren funtzionamendua ikusi ahalko da—:

   ../src/p5/main-deiak.c

   ```
   ./main_args_exekutagarria
   ./main_args_exekutagarria abc 123
   ./main_args_exekutagarria 10.0 Lisbeth Salander X
   ```

5.2.2. Ariketa

Ontziteria Urperatu! jokoaren `main` funtzioaren kodea aldatu, programaren deia jokalariaren izena erabiliz egin dadin. Horrela, jokoaren hasieran, jokalaria agurtu egingo du `puts` funtzioaren bidez.

6. praktika

Bi dimentsioko arrayak

Oholaren erabilera bi dimentsioko arrayen beharra erakusten hasia da. Praktika honetan, kodea bi dimentsioko arrayen bidez nola sinplifikatzen den ikusiko dugu.

6.1. Bi dimentsioko arrayetara aldatzea

Itsasontzien posizioa eta mota gordetzeko, bi dimentsioko array bat erabiliko dugu, eta beste bat egindako tiroak eta tiro horien emaitzak gordetzeko. Gainera, orain, horri esker, tamaina desberdinetako itsasontziak erabili ahal izango ditugu.

- Itsasontzien arrayak posizio batean (errenkada, zutabea) itsasontziaren tamaina adierazten duen zenbakia (betetzen dituen gelaxkak) izango du, bertan itsasontziren bat badago. Ordea, 0 bat izango du bertan itsasontzirik ez badago.
- Tiroen arrayak posizio batean -1 balioa izango du oraindik bertara tirorik egin ez bada; 0, tiroa egin bada eta bertan itsasontzirik ez badago; eta itsasontziaren tamaina adierazten duen zenbakia, baldin eta itsasontzirik badago.

Oholaren tamainarekin bat etorriko diren makro bi definituko ditugu: bat errenkada kopuruarekin eta beste bat zutabe kopuruarekin. Makro bi horiek erabilita, estatikoki erreserbaturiko arraya sortuko dugu. Datuak antolatzeko era berri horrek aldaketa hauek ekarriko ditu aurrerago eginiko kodean:

- `main` funtzioa
  - Oholaren tamaina definitzeko aldagaiek emaniko informazioa makroen bidez lortuko dugu orain.
  - Itsasontzien posizioa eta tamaina gordetzen zituzten aldagaiak bi dimentsioko array batez ordezkatuko dira. Array hori `int` motakoa izango da, eta oholaren tamaina izango du. Arrayaren posizio bakoitzeko, haren tamaina gordeko da, baldin eta itsasontzirik badago.

  - Tiroen jarraipena egiteko erabiltzen ziren aldagaiak ere bi dimentsioko array batez ordezkatuko dira. Array hori `int` motakoa izango da, eta oholaren tamaina bera izango du. Arrayaren posizio bakoitzean tirorik egin bada tiroaren emaitza gordeko da.

Main funtzioak jasan duen eraldaketa dela eta, haren ezaugarriak aldatu dira. Beraz, funtzioen deietan erabiliko diren sarrerako eta irteerako parametroak zein funtzioak berak ere moldatu beharko ditugu.

- `ArrayakHasieratu` funtzioak
  - Bi dimentsioko array biak hartzen ditu parametrotzat: itsasontzien posizioak eta egindako tiroak gordetzen dituena.
  - Orain ez du itzulera-baliorik bueltatuko.
  - Arrayak balio seguruekin hasieratzen ditu: itsasontzietako gelaxkek 0 balioa hartuko dute, eta tiroenek `-1`. Itsasontzien arrayan 0 bat egoteak esan nahi du posizio horretan ez dagoela itsasontzirik. Tiroen arrayan `-1` balioak adierazten du posizio horretan ez dela tirorik egin.
  - Bukatzeko, itsasontzi biren posizioa itsasontzien arrayan gordeko du. Bata horizontalean, 3 gelaxka betez, eta bestea bertikalean, 4 gelaxka betez. Horretarako, itsasontziak betetzen duen gelaxka bakoitzean zegoen 0a itsasontziaren tamaina adierazten duen zenbakiaz ordezkatu beharko da.

- `OholaMarraztu` funtzioa
  - Orain ez dira beharrezkoak tamaina adierazten duten parametroak, zuzenean errenkada eta zutabe kopuruak adierazten dituzten makroak erabiliko ditugu eta.
  - Eginiko tiroen berri izateko, nahikoa izango da dagokion bi dimentsioko arraya sarrerako parametrotzat hartzea.
  - Ohola bistaratzeko orduan, posizioz posizio tiroen arrayak duen balioari begiratu beharko zaio.
    * Haren balioa `-1` bada, oraindik ez da tirorik egin posizio horretara; beraz, zuriz bistaratuko da.
    * Bestela, arrayan gordetako balioa bistaratu beharko da. Balioa 0 izango da, tiroa egin, eta itsasontzirik ez bazegoen; bestela, itsasontziaren tamaina adierazten duen zenbakia bistaratuko da.

- `KoordenatuakIrakurri` funtzioa
  - Orain ez dira beharrezkoak tamaina adierazten duten parametroak, zeren zuzenean errenkada eta zutabe kopuruak adierazten dituzten makroak erabiliko baititugu.
  - Datuak baliozkoak diren egiaztatzeko prozesua eguneratu beharko da, zeren eta, orain azken errenkadari zegokion karakterea eta azken zutabeari zegokion zenbakia izan beharrean, errenkada eta zutabe kopuruak baitauzkagu.
  - Errenkada itzultzeko orduan, karakterea eman beharrean errenkadari dagokion zenbakizko balioa eman beharko da; dena den, aldagaiak `char` motakoa izaten jarraituko du. Zutabea lehen bezala itzultzen da.

- `TiroaEgiaztatu` funtzioa

  - Tiroaren errenkada eta zutabea zein itsasontzien posizioak eta egindako tiroak dituzten bi dimentsioko array biak pasatzen zaizkio sarrerako parametro gisa.
  - Lehenik eta behin, posizio horretan aurretik tiroa egin den edo ez aztertu beharko da. Horretarako, nahikoa da arrayko posizio horren balioa irakurtzea: aurretik tirorik egin bada (gelaxkak `-1` balioa ez badu), 0 balioa itzuli eta bukatu egin beharko da.
  - Tiroen arrayko posizioa erabili gabe bazegoen, itsasontzien arrayan posizio horretan dagoen balioa kopiatu behar da: 0, itsasontzirik ez badago, eta itsasontziaren tamaina adierazten duen zenbakia itsasontzirik badago.
  - Bukatzeko, balio hori itzuliko dio deitu dion funtzioari (`main`), asmatu duen edo ez jakin dezan, eta, horren ondorioz, biziak edo puntuak eguneratu daitezen.

6.2. Ontzidia erabat urperatzea

Orain arte ez dugu kontuan izan zer gertatzen den itsasontzi guztiak urperatzen direnean... Oraingoz, biziak agortu arte tiroak egiten jarraitzen utziko dio jokoak jokalariari. Hori konpontzeko, partida irabazitzat eman daitekeen begiratuko dugu.

- `ArrayakKonparatu` funtzioak
  - Parametro moduan itsasontzien eta eginiko tiroen posizioak dituzten bi dimentsioko array biak hartzen ditu.
  - `char` motako itzulera-balioa izango du: 0 balioa izango du, itsasontzirik oraindik geratzen bada, eta 1, ontzidi osoa urperatu bada.
  - Horretarako, egiaztatu beharko du itsasontzien posizioetan tiroen arrayak ez duela `-1` balioa.
- `main` funtzioak
  - Tiroa egiaztatu ondoren, `ArrayakKonparatu` funtzioari deitu beharko dio, partida bukatu den ala ez jakiteko.
  - Partida ontzidi guztia urperatu delako bukatzen bada, zoriontzeko mezua bistaratuko du.

Honelakoa izan daiteke, programaren irteera:

```
Ongi etorri "Ontzidia urperatu!"-ra
   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
A|  | 0|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
B|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
C|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
D|  |  |  | 3| 3| 3|  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
E|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
F|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0| 4| 0|  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
G|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
H|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Jo duzu!! 7 puntu dituzu.
Eta tiroa egiteko denbora maximoa 7 segundo izan da.

Zorionak, jokoa gaindituta!!!
```

7. praktika

Karaktere-kateak

Praktika honetan datu mota berri bat erabiliko dugu: karaktere-katea. C lengoaian datu mota hori erabiltzen da, karaktere berezi batekin —`NULL` edo `'\0'`— bukatzen den karakterez osaturiko array dimentsiobakar bat balitz bezala. C-ren oinarrizko liburutegiak karaktere-kateekin lan egiteko lagungarriak diren oinarrizko funtzioak ditu. Gai honetan, liburutegiko funtzio horiek erabiltzen ikasiko dugu.

7.1. Erabiltzaileari izena ematea

Jokoa hasten denean, erabiltzaileari izena eskatuko diogu. Hurrengo praktiketan jokalarien puntuazioei jarraipena egingo zaie. Informazio hori izateko, main funtzioa moldatu eta funtzio berri bat sortu beharko dugu.

- `main` funtzioa
  - Karaktere-kate bat sortzen du; haren tamaina maximoa makro baten bidez definituko dugu.
  - Jokoa hasi orduko, erabiltzailearen izena eskatuko da, ondoren azalduko den funtzioaren bidez. Izen hori sortu den aldagaian gorde beharko da.
  - Jokoa edozein arrazoigatik bukatzean (biziak agortzeagatik, ontzidia urperatzeagatik edo erabiltzaileak jokoa bukatzeko eskaera egiteagatik), erabiltzailearen izena eta lorturiko puntuazioa erakutsiko dituen mezu bat bistaratu beharko du.
- `IzenaEskatu` funtzioak
  - Sarrera/irteerako parametro gisa, makroan adieraziriko tamaina duen eta memorian gorderik dagoen karaktere-kate bati dagokion erakuslea erabiltzen du.
  - Erabiltzaileari mezu bat erakutsiko dio, izena galdetzeko; teklatu bidez sarturiko izena irakurri, eta erakusleak adierazitako kokagunean gorde beharko du.

7.2. Erabiltzailearen sarrera prozesatzea

Aurreko ataleko programa exekutatzean, ikusiko zenuten, erabiltzailearen izenaz gainera, errenkada-amaiera adierazten duen karakterea ere gordetzen dela. Horretaz gainera, erabiltzaileak izenik sartu ez badu, programak puntuazioari dagokion mezua izenik gabe bistaratzen du. Ondoren, arazo horiei eta beste batzuei konponbidea emango zaie.

- `KateaProzesatu` funtzioak
  - Sarrera/irteerako parametro gisa, `NULL` edo `'\0'` karakterearekin bukaturiko karaktere-kate bati loturiko erakuslea izango du.
  - Kate hori miatuko du, eta zuriz dauden hutsuneak eta tabuladoreak dakartzaten karaktereak (`' '` eta `'\t'`) ezabatuko ditu. Lorturiko emaitza emaniko helbide berean gordeko du; beraz, karaktere-katea berridatziko du.
- `IzenaEskatu` funtzioak
  - Erabiltzailearen sarrera irakurri ondoren, `KateaProzesatu` funtzioari deitu beharko dio.
  - Katea prozesatu eta gero, hutsik ote dagoen egiaztatu beharko du, eta, hala bada, `Ezezaguna` balioa gordeko du erabiltzailearen izen gisa.

8. praktika

Egiturak

Bazen garaia! Egiturak erabiltzeko aukera izango dugu hemendik aurrera. Egituren bidez posible izango dugu izaera ezberdinak izan arren osagai berdinari erreferentzia egiten dioten datuak multzokatzea.

8.1. Posizioa

Egiturak erabiltzean, oholean puntu bat kokatzeko eginkizuna duen egitura erraz batetik abiatuta egingo ditugu lehen urratsak.

- `Posizio` egitura
  - `char` motako lehen eremua oholeko errenkada adierazteko erabiliko dugu.
  - `int` motako bigarren eremuak oholaren zutabeari dagokion zenbakia gordeko du.
- `KoordenatuakIrakurri` funtzioak
  - Errenkadari eta zutabeari dagozkien parametroak jaso beharrean, `Posizio` motako egitura hartuko du erreferentziatzat, eta hantxe gordeko ditu teklatutik irakurritako koordenatuak.
- `TiroaEgiaztatu` funtzioak
  - Orain, sarrerako parametrotzat, tirorako errenkada eta zutabea barik, tiroaren kokapena barne duen `Posizio` motako egitura hartuko du.
- `main` funtzioa
  - Egitura berria erabiltzeko eta aldaketak jasan dituzten funtzioei era zuzenean deitzeko, eguneratu beharko dugu.

8.2. Jokalaria

Oraingo honetan, egitura berri bat sortuko dugu; hain zuzen, jokalariaren datu guztiak biltzen dituen egitura. Egitura hori etorkizunean oso erabilgarria izango da, erabiltzaile bakoitzaren puntuazioen jarraipena egiteko edo partidak gorde eta aurrerago berreskuratzeko.

- `Jokalari` egiturak
  - Lehen eremuan jokalariaren izena gordeko du. Izena karaktere-katetzat gordeko da, eta haren luzera maximoa finkaturik egongo da aurreko praktikan erabili dugun makroan zerrendatzen diren ezarpenen artean.
  - Bigarren eremuan jokalariaren puntuazioa `int` modura gordeko du.
  - Hirugarren eremuak jokalariaren bizi kopurua gordeko du.
  - Bukatzeko, laugarren eremuak jokalariak tiro egiteko eman duen denbora maximoa gordeko du.
- `JokalariaHasieratu` funtzioak
  - Aurreko praktikako `IzenaEskatu` funtzioa ordezkatuko du.
  - `Jokalari` motako egitura baterako erakuslea hartzen du sarrera/irteerako parametrotzat.
  - Teklatutik erabiltzailearen izena sartzeko eskatu eta aurreko praktikan zehazten zen erara prozesatutakoan, egitura barruko eremu egokian gorderik itzuliko du.
  - Jokalariaren puntuazioa 0ra hasieratuko du.
  - Jokalariaren bizi kopurua bizi kopuru maximora hasieratuko du.
- `main` funtzioa
  - Era egokian moldatu beharko dugu, lehen jokalariaren izenerako, puntuaziorako eta bizi kopururako erabiltzen genituen aldagaien ordez egitura berria erabil dezan.

8.3. Itsasontzia

Gauzak oraindik pixka bat gehiago korapilatuko ditugu: batetik, egitura bat beste egitura baten eremutzat erabiliko dugu; bestetik, egitura-array batekin egingo dugu lan. Horretarako, itsasontziaren posizioa gorde eta gure ontzidia jokoaren hasieran marrazteko erabiliko dugun egitura sortuko dugu.

- `Itsasontzia` egiturak
  - `Posizio` motako eremuak itsasontziaren mutur bati dagokion posizioa gordeko du.
  - Jarraian, `char` motako eremua izango du, itsasontziaren norabidea adierazteko. Eremu horrek hartuko dituen balioak hauek izango dira hurrenez hurren: `'H'`, itsasontzia horizontalean badago; `'B'` itsasontzia bertikalean badago; edo `'D'`, itsasontzia diagonalean badago.

  - Bukatzeko, `int` motako eremuan, itsasontziaren luzera zehaztuko du.

- `ItsasontziaMarraztu` Funtzioak
  - Sarrerako bi parametro hartuko ditu: `Itsasontzia` motako egitura bat —haren bidez marraztu beharreko itsasontziari dagozkion datuak jasoko ditu— ; eta itsasontzien kokapena gordeko duen bi dimentsioko arraya.
  - Parametrotzat hartu den itsasontzia marraztuko du bi dimentsioko arrayaren laukietan.

Array bien baliozko hasieraketarako erabili duzun kodea gorde, eta hasierako itsasontziak marrazteko erabiltzen zenuen kodea ezabatu.

- `ArraiakHasieratu` funtzioak
  - `Itsasontzia` motako egituren arraya sortu eta hasieratuko du; bertan, hiru itsasontzi kokatuko ditu: 3 luzerako bat, horizontalki; 5 luzerako bat, bertikalki; eta 4 luzerako bat, diagonalki.
  - Begizta baten bidez, array horren ibilaldia egingo du; hiru itsasontziak marrazteko `ItsasontziaMarraztu` funtzioari behar bezala deituko dio.

9. praktika

Fitxategi bitarrak

Fitxategiak iraunkor bihurtzen du programa exekutatu bitartean erabiltzeak sortu duen informazioa, hau da, informazio hori exekuzio batetik bestera ez galtzeko aukera ematen digu. Hori dela eta, esaterako, informazio hura programa berriro ere exekutatzean berreskura daiteke. Gure kasuan, ezaugarri hori erabiliko dugu, besteak beste, bukatu gabeko partidak gordetzeko eta etorkizunean birkargatu eta jolasten jarraitzeko.

9.1. Partida gorde

Esan eta egin! Fitxategiak erabiltzen trebatu garenean, erabiltzaileak oraindik bukatu gabeko partidatik ateratzeko agindua eman eta partidaren uneko egoera gordeko dugu. Programa berriz ere exekutatzen denean, automatikoki, partida hura berreskuratu eta abian jarriko da.

- `PartidaGorde` funtzioak
  - `Jokalari` motako egitura batera zuzenduriko erakuslea, itsasontzien arrayra zuzenduriko erakuslea eta egindako tiroen arrayra zuzenduriko erakuslea hartzen ditu sarrerako datutzat.
  - `int` motako balioa itzultzen du: 0, partida gordetzean akatsik gertatu bada; 1, eragiketa guztiak zuzen gauzatu badira.
  - Fitxategi bitarra zabaldu eta hiru parametroen bidez hartu dituen datuak fitxategian gordeko ditu. Noizbait eragiketa horiek exekutatzean akatsik gertatuz gero, akatsmezua pantailaratuko eta 0 balioa itzuliko du.
  - Fitxategia itxiko du.

- `PartidaKargatu` funtzioak
  - `Jokalari` motako egitura batera zuzenduriko erakuslea, itsasontzien arrayra zuzenduriko erakuslea eta egindako tiroen arrayra zuzenduriko erakuslea hartzen ditu sarrerako datutzat.
  - `int` motako balioa itzultzen du: 0, partida kargatzeko orduan akatsik gertatu bada; 1, eragiketa guztiak zuzen gauzatu badira.
  - Fitxategi bitarra zabaldu eta fitxategian gorderik dauden datuak hiru parametroetan gordeko ditu. Noizbait eragiketa horiek exekutatzean akatsik gertatuz gero, akatsmezua pantailaratuko eta 0 balioa itzuliko du.
  - Fitxategia itxiko du.

Jokoaren bukaeraren kudeaketak gero eta lan gehiago duenez, hura `main` funtziotik kendu, eta kodea funtzio propio batera, honetara, eramango dugu.

- `Bukatu` funtzioak
  - `Jokalari` motako egitura batera zuzenduriko erakuslea, itsasontzien arrayra zuzenduriko erakuslea eta egindako tiroen arrayra zuzenduriko erakuslea hartzen ditu sarrerako datutzat.
  - Jokoa bukatzeko dauden hiru aukerak aztertuko ditu (jokoa irabazi dugu, *game over* eta lagatako partida), eta gertatutakoaren araberako mezua erakutsiko da pantailan.
  - Gainera, hemendik aurrera, erabiltzaileak berak eskatzen badu jokoa amaitzeko, gerora jokoa berreskuratzeko aukera izan dezan, partida gorde nahi ote duen galdetuko zaio. Baiezko erantzuna hartuz gero, `PartidaGorde` funtzioari deituko dio.
  - Funtzioak 0 itzuliko du akatsik izan bada, eta 1, eragiketa guztiak zuzenak izan badira.

Jokoa hasieratzeko egin beharreko lanak gero eta gehiago direnez, `main`-etik atera eta Hasi funtzio honetara ekarriko ditugu.

- `Hasi` funtzioak
  - `Jokalari` motako egitura batera zuzenduriko erakuslea, itsasontzien arrayra zuzenduriko erakuslea eta egindako tiroen arrayra zuzenduriko erakuslea hartzen ditu sarrerako datutzat.
  - Gordetako partida bati dagokion fitxategirik ote dagoen egiaztatuko du:
    - Halakorik ez badago, orain arte egin den modura, `ArrayakHasieratu` eta `JokalariaHasieratu` funtzioei deituko die.
    - Bukatu gabeko partida topatuz gero, `PartidaKargatu` funtzioari deituko dio.
  - Funtzionamendu zuzena izan badu, 1 itzuliko du, eta 0, akatsen bat gertatu bada.
- `main` funtzioa
  - `Hasi` eta `Bukatu` funtzioak egoki erabiltzeko eta itzultzen dituzten balioen zuzentasuna aztertzeko, eguneratu beharko dugu.

9.2. Puntuazioa gogoratzea

Atal honetan, jokoan izandako puntuazio hoberenen zerrenda eguneratua sortuko da. Horretarako, bukatutako partida bakoitzeko puntuazioak gordeko dituen fitxategi bitarra berrituko dugu.

Sortu puntuazioen erregistroaren tamaina maximoa zein den ezarriko duen makroa (5 puntuazio) eta puntuazioen erregistroa gordeko duen fitxategiaren izena zehaztuko digun makroa. Era horretara, makro horiek funtzio guztietan erabiliko ditugu programan, balioak behin eta berriro errepikatu beharrik izan gabe.

- `PuntuazioaGorde` funtzioak
  - Puntuazio guztiak gordetzen dituen `Jokalari` motako array baterako erakuslea hartzen du sarrerako parametrotzat.
  - Idazkera bitarra egiteko, puntuazioen fitxategia zabalduko du. Horretarako, fitxategi horren bertsio zaharra ordezkatuko du, edo fitxategi berria sortuko du aurretiaz izen horretakorik ez bazegoen.
  - Puntuazioak banan-banan gordeko ditu, 0 puntura edo makroan adierazitako osagai kopuru maximora heldu arte.
  - Fitxategia itxiko du.
- `PuntuazioaKargatu` funtzioak
  - Aldez aurretik erreserbaturik dagoen eta puntuazioak gordetzeko erabiliko den `Jokalari` motako array baterako erakuslea hartzen du sarrerako parametrotzat.
  - Puntuazioen fitxategia irakurketa bitarra egiteko zabaltzen du.
  - Fitxategirik ez bazegoen, arrayaren lehen osagaiari dagokion puntuazioaren eremua 0 baliora hasieratuko du. Era horretan, puntuazioen erregistroak irakurtzeari noiz utzi jakingo dugu.
  - Fitxategiko puntuazioak irakurriko ditu, fitxategiaren amaiera edo erregistro kopuru maximoa irakurri arte. Fitxategiaren amaiera erregistro kopuru maximoa bete baino lehen aurkitu bada, baliozkoa izan den azken osagaiaren ostean datorren osagaiaren puntuazioaren eremua 0 balioaz hasieratuko dugu. Hala jakingo dugu osagai hori eta hurrengoak ez direla kontuan hartu beharrekoak.
  - Fitxategia itxiko du.
- `PuntuazioaGaurkotu` funtzioak
  - Une horretan ari den jokalariaren datuak dituen `Jokalari` motako datu baterako erakuslea eta puntuazioen arrayrako erakuslea hartzen ditu sarrerako parametrotzat.
  - Egungo puntuazioa dagokion posizioan gordeko du. Horretarako:
    * Jokalariak 0 puntu baditu, funtzioa inolako lanik egin gabe bukatuko da; izan ere, punturik gabeko jokalariak ez dira erregistratuko, zeren 0 puntuazioa erabiliko baitugu markatzeko arrayaren osagai hori zein jarraian datozenak erabili ez diren puntuazio-erregistroak direla.

        * Jokalariaren puntuazioa beste bat bada, puntuazioen taulan dagokion kokapena bilatuko dugu; gogoan izan lehenik puntuaziorik hoberenak/handienak eta amaieran txarrenak daudela.
            · Izandako puntuazioa taulan daudenekin alderatuz txarrena bada, taularen amaieran sartuko dugu, baldin eta horretarako lekurik badago (makroan adierazitako erregistro kopuru maximoaren arabera).
            · Egungo puntuazioa erdiko puntu batean sartu behar bada, hura baino txarragoak diren puntuazioak mugitu beharko ditugu eta, behar izanez gero—beste puntuazio bat sartzeko lekurik ez badago—txarrena baztertu.

- `PuntuazioaBistaratu` funtzioak
    - Puntuazioetara zuzenduriko erakuslea hartzen du parametrotzat.
    - Puntuazioen arrayaren ibilaldia egingo du, jokalariaren izena eta izandako puntuazioa ordenan bistaratuz.

Funtzioa eguneratu beharko da, jokoa bukatu baino lehen puntuazioen erregistroa berritu eta erakuts dezan.

- `Bukatu` funtzioak
    - Horretarako sortuberriak diren funtzioei deituko die:
      `PuntuazioaKargatu`,
      `PuntuazioaGaurkotu`,
      `PuntuazioaGorde` eta bukatzeko
      `PuntuazioaBistaratu`.

Hauxe duzue programaren irteeraren adibide bat:

```
Ongi etorri "Ontzidia Urperatu!"-ra

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
A|  | 0|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
B|  |  |  |  |  | 5|  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
C|  |  |  |  |  | 5|  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
D|  |  |  |  |  | 5|  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
E|  |  |  |  |  | 5|  |  |  |  | 4|  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
F|  |  |  |  |  | 5|  |  |  |  |  | 4|  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
G|  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
H|  |  |  |  |  |  |  |  |  |  |  |  |  | 4|  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
I|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
J|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Ura!!, oraindik 0 aukera/bizi dituzu.
Eta tiro egiteko denbora maximoa 5 segundo izan da.

Gorka-ren puntuazioa: 9


--- Game Over ---

Puntuazioak:
Gorka   9
aaa     6
Gorka   5
```

10. praktika

Testu-fitxategiak

Testu-fitxategiek aukera ematen digute informazioa formatu irakurterraz eta ulergarria den formatu batean gordetzeko.

Praktika honetan, gure aplikazioaren parte ez den testu-editore baten bidez sorturiko pantailak kargatzeko erabiliko ditugu testu-fitxategiak.

10.1. Pantaila sortzea

Testu-editore baten laguntzaz[1], jokoaren oholaren tamainaren arabera, sortu zenbakiz osatutako fitxategi bat, zenbakiak errenkada eta zutabetan antolatuak eta zuriunez banatuak dituena.

Segidan, aurrekoaren eredu moduko fitxategi bat erakusten da:

```
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 5 0 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5 0 0
0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0
3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

[1] OpenOffice moduko testu-prozesadorea erabiliz gero, ziurtatu TXT formatuan eta ez ODT formatuan gorde duzula fitxategia.

10.2. Pantaila kargatzea

Gure programa moldatuko dugu orain, erabiltzailearen aginduari jarraituz hartzen duen parametroa kargatu beharreko pantaila gordetzen duen fitxategiaren izena izan dadin. Programa inolako parametrorik gabe exekutatuz gero, funtzionamenduak orain artekoa izaten jarraituko du.

- `PantailaKargatu` Funtzioak
  - Sarrerako parametrotzat itsasontzien kokapena duen bi dimentsioko arrayrako erakuslea eta pantaila gordetzeko erabili den fitxategiaren izena zein den zehazten duen karaktere-katea hartzen ditu.
  - Eragiketa guztiak zuzen garatu badira, 1 balioa itzuliko du; 0, berriz, akatsen bat gertatu bada.
  - Testuen irakurketa egiteko adierazi zaion fitxategia irekiko du. Akatsik badago, horren berri eman beharko du, dagokion akats-kodea itzuliz.
  - Arrayko itsasontzien posizioak fitxategiak dituen zenbakien bidez lortuko dira.
  - Fitxategia itxi eta horren berri eta horretarako erabili den fitxategiaren izena adierazten duen mezua aterako du pantailan.
- `Hasieratu` Funtzioa
  - Kasu honetarako, sarrerako parametro bat gehitu beharko zaio; kargatu beharreko fitxategiaren izena alegia.
  - Bukatu gabeko partidarik badago edo hartzen den parametroa `NULL` bada, ez da pantaila berririk erabiliko.
  - Bukatu gabeko partidarik ez badago eta fitxategi-izen onargarri bat adierazi bada, ordea, `PantailaKargatu` funtzioari deituko dio, eta horretarako ezinbestekoak zaizkion parametroez hornituko da.
- `main` Funtzioak
  - Hartzen dituen parametroak aztertu beharko ditu: horretarako, prototipoa era egokian moldatu beharko dugu.
  - `Hasieratu` funtzioa deitzean, lehen parametro modura jaso duen fitxategiaren izena zehaztu beharko du; parametrorik hartzen ez duen kasuan, berriz, `NULL` balioaz egingo du deia.

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Programazioaren Oinarriak

Javako Praktikak

Telekomunikazio Ingeniaritza Teknikoa - 1. maila

Bilbo, 2010eko iraila

Programazioaren Oinarriak: Javako Praktikak

Laburpena

Praktika-bilduma honen helburua zera dugu: ikasleek *Java erabiliz klase teorikoetan lantzen diren Objektuetan Oinarritutako Programazioari (OOP) buruzko ezagutzak sendotzea*.

Bilduma honetan agertzen diren hamar praktikak bukatutakoan, ikasleak kontsolako joko erraz batzuen artean aukeratu eta jostatzeko beta emango dion aplikazio bat izango du Java programazio-lengoaian idatzia.

Azken aplikazio horretara iristeko, ikasleak praktikaz praktika teoriako klaseetan landutako kontzeptuak erabiliko ditu, eta, programa erraz eta labur batetik abiaturik, azken kodea osatuko du. Proposaturiko hobekuntza berri bakoitzaren helburua zera da, aplikazioaren aurreko bertsioaren aldean dakarren hobekuntza zein den azpimarratzea.

GOMENDATURIKO JOKAERA: Lehenengo praktikan izan ezik, gainerakoetan, praktika batetik hurrengo praktikara pasatu eta aurreko praktika horretan garatutakoa ez galtzeko, praktika berri bakoitzaren hasieran ikasleak aurreko praktikan erabili dituen *fitxategi guztien kopia* egin beharko du bere direktorioan, kopia hori eta bertako fitxategiak funtzionaltasun berriak gauzatzeko erabiliko baititu.

Gaien Aurkibidea

0. Praktika

Ingurunea ezagutzea

Praktika honetan ikasleak lehen hartu-emana izango du programazio-ingurunearekin, dela interfaze grafiko bidez (Eclipse), dela agindu-lerroa erabiliz. Horretarako, programa sinple bat eraikiko, konpilatuko eta exekutatuko da.

0.1. *Workspace*-a

Lehenik eta behin, *user1* erabiltzailearentzako `home` izango den karpetan, bakoitzak bere izena izango duen karpeta bat sortu beharko du. Hortik aurrera, karpeta hori erabiliko da praktika guztiak gordetzeko, eta, aurrerantzean, *workspace* deitua izango da.

Biziki gomendatzen dizuegu, klase bakoitza bukatutakoan zuen *workspace* barruan dagoen guztia USB memoria batean kopiatzea —backup modura izan dezazuen—. Izan ere, Kalkulu Zentroetako ordenagailuak eta haietan dauden instalazioak noiznahi eta aldez aurretik jakinarazi gabe berrabiarazten baitira.

0.2. "Kaixo Java" kontsola bidez egitea

Testu-editore baten laguntzaz[1], idatzi kode hau:

```
public class KaixoJava {
    public static void main( String args[] ) {
        System.out.println( "Kaixo Java!" );
    }
}
```

[1] OpenOffice-ren moduko testu-prozesadorea erabiliz gero, ziurtatu fitxategia TXT formatuan gordetzen duzula, eta ez formatu bitar batean.

Gorde ezazu iturri-fitxategia `Praktika0-1` karpetan —norberaren *workspace* karpetan, zuen *home* izango den horretan—, `KaixoJava.java` izenaz.

Hurrengo urratsa programa konpilatzea izango da. Horretarako, kontsola zabaldu eta, iturri-kodea duen karpetara joango gara —*workspace* barruan dagoen `Praktika0-1`—. Hori egiteko erabili beharreko aginduak hauek izango dira:

```
$ cd zure_izena
$ cd Praktika0-1
```

Karpeta barruan dauden fitxategien zerrenda agindu honekin azter dezakegu:

```
$ ls
```

Karpeta honetan kokaturik, java-konpiladoreari deituko zaio; parametro modura konpilatu nahi den fitxategiaren izena pasatuz:

```
$ javac KaixoJava.java
```

Berriz ere, `ls` agindua exekutatuz gero, ikusiko dugu `KaixoJava.class` fitxategi bat sortu dela. Fitxategi horrek, java-ko *bytecode*-a—java-ko makina birtual batek interpretatu dezakeen kodea—izango du bere barnean. Horri deitzeko, agindu hau erabili beharko da:

```
$ java KaixoJava
```

Ikus daitekeen moduan, pasatzen zaion parametroa ez da Java-fitxategiaren izena, `main` metodoak duen klasearen izena baizik. Programa hori exekutatzean, hau bistaratuko da:

```
Kaixo Java!
```

0.3. "Kaixo Java" Eclipse erabiliz egitea

Pareko prozesua egingo dugu, baina, oraingo honetan, Eclipse erabiliz.

Hasteko, Eclipseri gure *workspace*-a zein den adierazi beharko diogu. Hori, programak duen `File/Switch Workspace ...` aukeraren bidez egingo da; hor, erabiliko dugun direktorioa `/home/user1/zure_izena` dela adieraziko dugu. Aurreko hori praktika bakoitza hasterakoan egin beharrekoa izango da, ordenagailua erabili dugun azken alditik hona beste norbaitek Eclipse erabili eta bere *workspace*-a jarri ahal izan baitu, eta gurea ordezkatu.

Orain, proiektu berri bat sortzeko prest gaude. Horretarako, `File/New Project` hautatuko dugu, `Java Project` bat sortu nahi dugula adieraziko dugu eta `Praktika0-2` izena emango diogu. Eclipsek proiektu bakoitza karpeta batean gordetzen du—*workspace*-aren barruan—.

Proiektuaren zuhaitzean, eskuineko botoiarekin klik egin eta klase berri bat sortzea hautatuko dugu. `KaixoJava` deituko diogu klase berri horri, eta automatikoki iturri-fitxategi bat sortu dela ikusiko dugu. Fitxategi horretan aurrerago erakutsiriko kodea idatzi, eta fitxategia gordeko dugu.

Azkenik, `Run` menura joan, eta `Run As/Java Application` aukeraren bidez, programa konpilatuko eta exekutatuko dugu. Exekuzioaren emaitza Eclipseko `Console` leihoan bistaratuko da.

Hortik aurrera, ikasleak nahiago duen modura egin ahal izango ditu praktikak.

0.4. Teklatu bidez sarturiko sarrera-datuen irakurtzea

Hurrengo praktiketan egingo diren programek teklatu bidez sarturiko sarrera-datuak beharko dituzte. Hori egiteko beharrezkoak diren kontzeptuak teorian emango ez direnez, kasu horietan, irakasleak sorturiko kodea erabili beharko duzue. Kode horrek `Teklatua.KaraktereaIrakurri()`, `Teklatua.ZenbakiosoaIrakurri()` eta `Teklatua.KateaIrak` metodoak definituko ditu. Metodo horiek, erabiltzaileak teklatu bidez sarturikoaren arabera, hurrenez hurren, `char`, `int` edo `String` bat itzuliko dute.

Kopia ezazu kode hau `Teklatua.java` izeneko fitxategi batean, eta fitxategia, sortu beharreko `Praktika0-3` direktorio berri baten barruan gorde:

```
import java.io.*;

public class Teklatua {
    public static char KaraktereaIrakurri() {
        char ch;

        try {
            ch = KateaIrakurri().charAt(0);
        } catch( Exception e ) {
            ch = 0;
        }

        return ch;
    }

    public static String KateaIrakurri() {
```

```
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
        String str;

        try {
            str = br.readLine();
        } catch( Exception e ) {
            str = "";
        }

        return str;
    }

    public static int ZenbakiosoaIrakurri() {
        int num;

        try {
            num = Integer.parseInt( KateaIrakurri().trim() );
        } catch( Exception e ) {
            num = 0;
        }

        return num;
    }
}
```

Teklatuaren funtzionamendua zuzena den ala ez baieztatzeko, itsats ezazue kode hau direktorio berean sortuko duzuen `KaixoTeklatua.java` izeneko fitxategi batean:

```
1   public class KaixoTeklatua {
2       public static void main( String args[] ) {
3           String izena;
4           int adina;
5
6           System.out.print( "Esan zure izena: " );
7           izena = Teklatua.KateaIrakurri();
8
9           System.out.print( "Esan zure adina: " );
10          adina = Teklatua.ZenbakiosoaIrakurri();
11
12          System.out.println( "Kaixo " + izena +
13                              ", " + adina + " urte dituzu." );
14      }
15  }
```

Saia zaitezte fitxategi biak kontsola bidez eta Eclipsen sorturiko proiektu berri bat

erabiliz konpilatzen.

0.5. Aplikazioak araztea

Eclipsek aplikazio-araztea interfaze grafiko bidez egitea errazten du. Eclipse bidez Javaren kode-arazketa era zuzenean egin dadin, aplikazioen arazketa baimentzen duen JRE bat erabili beharko dugu. Beharrezko den JRE hori erabiltzen ari garela baieztatzeko, `Window/Preferences/Java/Installed JREs`-n sartu eta SUN-en JRE aktibatua dagoen egiaztatu beharko dugu. Aukera hori egon ere ez badago, ondorengo *path* hau erabiliz gehituko diogu: `/usr/lib/jvm/java-6-sun-1.6.0.03/jre`.

Ondoren, `Teklatu.java` fitxategian, *breakpoint* bi jarriko ditugu: bat 6. lerroan, eta bestea 9.ean. *breakpoint* bat jartzeko, lerroaren ezker hegalean jarri, arratoiaren eskuinaldeko botoian klik egin, eta `Toggle Breakpoint` aukeratu besterik ez da egin behar —lerroaren ertzean ezkerraldeko botoiaz bitan klik eginez gero, gauza bera lortzen da—. Prozesu berdina egin beharko da aurretiaz jarritako *breakpoint* bat kentzeko.

Behin *breakpoint*-ak jarrita, arazteko, `Run` menura joan, eta haren barnean duen `Debug As/Java Application` aukeratu beharko da. Programa 6. lerrora heltzean gelditukо da; ondoren, F6 tekla sakatuz gero, instrukzio hori exekutatu eta prozesua hurrengo lerroan etenda gelditukо da. Jarraitzeko, F8 sakatuz gero, programa hurrengo *breakpoint*-era iritsi arte exekutatuko da.

Aurreko prozesua berregin, oraingo honetan F6-ren ordez F5 erabiliz. Eztabaidatu itzazue ikusi dituzuen desberdintasunak.

1. Praktika

Klase eta objektuak

Lehenengo praktika honetan, hemendik aurrerako praktika guztietan egingo dugun lanarekin eratuko den programarekin hasiko gara. Praktika zehatz honen helburua zera da, ikasleak klase eta objektu kontzeptuak erabiltzen hastea eta haiek erabiltzen trebatzea.

1.1. Jokoa klasea

Hasteko, ezaugarri hauek izango dituen `Jokoa` izeneko klase bat eratuko da:

- Atributuak
  - Atributu publiko gisa zenbaki oso bat izango du; balio horrek jokalariari geratzen zaion bizi kopurua gordeko du.
- Metodoak
  - Metodo eraikitzaile bat izango du; osoa motakoa izan, eta jokalariak duen bizi kopuruaren hasierako balioa gordeko duen sarrerako parametroa jasoko du.
  - `BiziKopuruaBistaratu` izena hartuko duen metodoa izango du; metodo horrek jokoak dirauen bitartean jokalariari gelditzen zaion bizi kopurua bistaratuko du.
  - Metodo horietaz gainera, honako kode hau bilduko duen `main` metodoa ere izango du:
    * `Jokoa` klaseko instantzia bat sortuko du; horretarako, hasierako bizi kopuruak 5 balioa hartuko du.
    * Sortu den objektuaren `BiziKopuruaBistaratu` metodoari deituko dio.
    * Bizi kopurua gordetzen duen atributuaren balioari bat kendu, eta ondoren, `BiziKopuruaBistaratu` metodoari berriz deituko dio.

    * `Jokoa` klaseko beste instantzia bat sortuko du; aurrekoan bezala, hasierako bizi kopuruak 5 balioa hartuko du.
    * Objektu berri horren `BiziKopuruaBistaratu` metodoari deitzen dio, eta ondoren, lehenengo objektuko metodo berdinari.

1.2. Atributuak ezkutatzea

Aurreko ariketan, ez diogu arretarik jarri `Jokoa` klaseko bizi-atributua aldatzea posible izan dadin egin beharrekoari. Hori ekiditeko, ariketa honetan, baimenduriko atxikipena jarriko zaio atributu horri. Horretarako, aldaketa hauek gauzatu beharko ditugu:

- Atributuak
  - Bizi kopurua duen atributuak beste edozein klaserentzat ezkutukoa izan beharko du. Beraz, hortik aurrera, atributu hori aldatu ahal izateko, aurrerago azalduko diren metodo bi sortu beharko dira.
  - Beste atributu bat sortu beharko da —hori ere pribatua—; horrek eraikitzaileari hasiera batean pasatu zitzaion bizi kopurua gordeko du. Atributu hori jokoa berrabiatu ahal izateko erabiliko da.
  - Ondoren, zenbaki oso bat izango den eta markarik onena gordeko duen beste atributu pribatu bat sortu beharko da. Atributu horrek aurrekoekiko desberdintasun nabarmen bat izango du; kasu honetan, instantzia-atributu bat izan beharrean, klase-atributu bat izango da. Beraz, eratuko diren joko guztientzat komuna izango da eta haren hasieratze-balioa 0 izango da.
- Metodoak
  - `BiziaKendu` metodoa gehitu; batean gutxituko du jokalariaren bizi kopurua eta bizirik geratzen zaion ala ez adieraziko duen `boolean` balio bat itzuliko du. Jokalariak biziak agortu baldin baditu, `Jokoa bukatu da!` mezua bistaratuko da.
  - `PartidaBerrasi` metodoa gehitu; horretan, bizi kopurua duen atributuari, eraikitzaileari deitu zitzaionean pasatutako bizi kopurua esleituko zaio. Hori egiteko, jadanik eraturik dagoen atributu berria erabiliko da.
  - Markarik onena duen balioa jokalariari gelditzen zaion bizi kopuruarekin konparatuko duen `MarkaGaurkotu` metodoa sortu behar da.
    * Gelditzen zaion bizi kopurua marka onenaren berdina bada, marka lortu dela adieraziko duen mezu bat bistaratu beharko da.
    * Gelditzen zaion bizi kopurua marka onena baino handiagoa bada, marka eguneratuko da, eta marka hobetu dela eta lortutako balio berria adieraziko duen mezu bat bistaratuko da.
    * Gelditzen zaion bizi kopurua marka onena baino txikiagoa bada, metodoak ez du ezer egingo.

Atributua ezkutatu den frogatzeko, `main` funtzioa `Aplikazioa` deituko diogun klase berri batean —direktorio bereko `Aplikazioa.java` fitxategian— jarriko dugu.

- Hori eginda, bizi kopurua duen atributua aldatzen saiatzen zaretenean, konpiladoreak errore-mezu bat itzultzen dizuela baieztatu beharko duzue.
- Errore hori konpontzeko, aldaketa hauek gauzatu beharko dira:
  - `Jokoa` klaseko instantzia bateko `BiziaKendu` metodoari deitu; behin hori eginda, `BiziKopuruaBistaratu` metodoari deitu.
  - Ondoren, `PartidaBerrasi` metodoari deitu, eta `BiziKopuruaBistaratu` metodoari berriz ere.
  - `Jokoa` klaseko lehenengo instantziako `MarkaGaurkotu` metodoari deitu, eta jarraian, bigarren instantziako metodo berdinari.
- Aurreko exekuzioaren ondorioz `agertzen diren mezuak azaldu`.

2. Praktika

Herentzia eta polimorfismoa

Praktika honetan herentzia eta polimorfismo kontzeptuak erabiliko ditugu. Teoriako klaseetan ikusi den moduan, herentzia bidez, klase berriek aurretiaz garaturiko beste klase batzuetako kodea berrerabil dezakete. Polimorfismoa, berriz, metodo bati kasuan-kasuan parametro desberdinak erabiliz deitzea posible egiten duen ezaugarria da. Poliformismoaren kasuan, kasu bakoitzean metodo horri deitzen dion funtzioaren ezaugarrien edo beharrizanen arabera finkatuko dira parametroak.

Horrez gain, azken aplikazioari itxura ematen hasiko gara.

2.1. 1. ariketa

Ariketa honetan joko bat sortuko dugu, zeinean, erabiltzaileak[1] programak definitzen duen zenbaki bat asmatu beharko baitu. Horretarako sortuko den klase bakoitzari dagokion kodea, Java-fitxategi desberdin batean gorde beharko da —klasearen izen berbera duen fitxategi batean—.

- `Jokoa` klasea
  - `Jokatu` metodo abstraktu bat gauzatuko du; ez du sarrerako parametrorik izango, eta eratorritako klaseek inplementatua izango da.
  - Orain, `Jokoa` klasea abstraktua izango da; horrek ezinezko egingo du beraren instantzia sortzea.
  - `main` metodoa klase horretatik kanpo atera beharko da.
- `ZenbakiaAsmatuJokoa` klasea
  - `Jokoa` klasetik eratorria.

[1] "'Ingurunea ezagutu"' praktikan kopiatu dugun klaseetariko bat erabiliz, teklatutik zenbaki oso bat irakurriko da.

  - Osoa motako sarrera-parametro bi dituen metodo eraikitzaile bat izango du. Lehen parametroan bizi kopurua gorde, eta oinarrizko klasearen eraikitzaileari pasatuko zaio. Bigarren parametroa asmatu beharreko zenbakia izango da; haren balioak 0tik 10era tartean egon beharko du.
  - Oinarrizko klaseko `Jokatu` metodoa inplementatuko du:
    * Heredatu duen `PartidaBerrasi` metodoari deituko dio.
    * Erabiltzaileari mezu bat erakutsiko dio, 0-10 bitarteko zenbaki bat asmatu dezan eskatzeko.
    * Teklatutik zenbaki oso bat irakurri, eta programatzaileak jarririko balioarekin konparatuko du:
      · Balio biak berdinak badira, `Asmatu duzu!!!` dioen mezu bat bistaratuko du, `MarkaGaurkotu` metodoari deituko dio, eta metodotik aterako da.
      · Desberdinak badira, heredatu duen `BiziaKendu` metodoari deitu, eta hura exekutatuko du:
      · `BiziaKendu` metodoak `true` itzultzen badu, jokalariak oraindik biziak ditu, beraz, asmatu beharreko zenbakia teklatutik sarturikoa baino handiagoa ala txikiagoa den adierazi eta berriz saiatzea eskatuko duen mezu bat bistaratuko da.
      · `BiziaKendu` metodoak `false` itzultzen badu, jokalariari bizirik geratzen ez zaiola adierazten du. Kasu horretan, `Jokatu` metodotik atera beharko du.

- `Aplikazioa` klasea
  - `main` metodo bat izango du; metodo horrek sortu den `ZenbakiaAsmatuJokoa` klasearen instantzia bat sortuko du, eta `Jokatu` metodoari deituko dio.

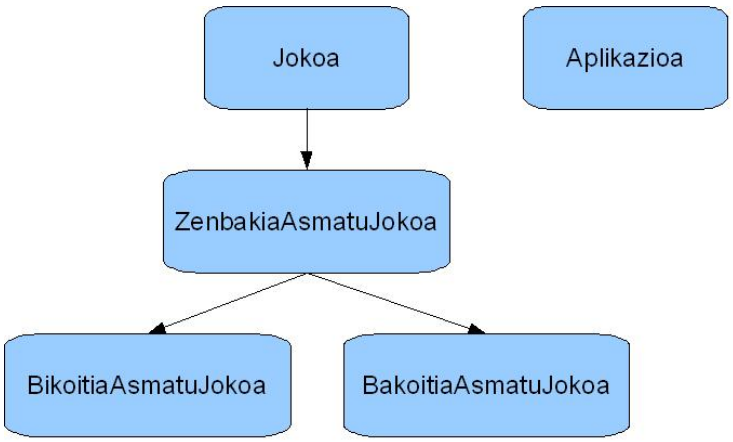
2.2. 2. ariketa

Aurreko jokoa abiapuntutzat hartuta, joko berri bi eratuko dira; batak zenbaki bakoitiak asmatzeko balioko du, eta besteak zenbaki bikoitiak asmatzeko.

- `ZenbakiaAsmatuJokoa` klasea
  - `ZenbakiaBalioztatu` metodo berri bat sortu beharko da. Metodo horrek, sarrera-parametrotzat erabiltzaileak sarturiko zenbakia hartu eta beti (kasu honetan) `true` balioa izango duen `boolean` bat itzuliko du.
  - `Jokatu` metodoan, `ZenbakiaBalioztatu` metodoak `false` balioa itzuliz gero, teklatutik beste zenbaki bat sartzea eskatuko zaio erabiltzaileari. Horrez gain, kasu horretan ez da bizirik kenduko.

- `BikoitiaAsmatuJokoa` klasea
  - `ZenbakiaAsmatuJokoa` klasetik eratorria.
  - `ZenbakiaBalioztatu` metodoa birdefinitzen du. Kasu honetan,`true` balioa itzuliko du zenbakia bikoitia bada; zenbakia bakoitia izanez gero, pantailan errore-mezu bat erakutsiko eta `false` balioa itzuliko du.
- `BakoitiaAsmatuJokoa` klasea
  - `ZenbakiaAsmatuJokoa` klasetik eratorria.
  - `ZenbakiaBalioztatu` metodoa birdefinitzen du. Kasu honetan, `true` balioa itzuliko du zenbakia bakoitia bada; zenbakia bikoitia izanez gero, pantailan errore-mezu bat erakutsiko eta `false` balioa itzuliko du.
- `Aplikazioa` klasea
  - `main` metodoan, praktikan sorturiko hiru jokoetariko —`ZenbakiaAsmatuJokoa`, `BikoitiaAsmatuJokoa` eta `BakoitiaAsmatuJokoa`— bakoitzaren instantzia bat sortuko da. Joko bakoitzaren hasierako bizi kopurua 3 izango da, eta, asmatzeko zenbaki gisa: edozein zenbaki, zenbaki bikoiti bat eta zenbaki bakoiti bat, hurrenez hurren. Eta denek 0tik 10era artekoak izan beharko dute.
  - Horrez gain, metodo horretan, hiru instantzietako bakoitzaren `Jokatu` metodoari deituko dio.

Sorturiko klaseen hierarkia irudi honetan ikus daiteke:



3. Praktika

Interfaze eta arrayak

Praktika honetan interfaze kontzeptua erabili eta interfaze-arrayak sortuko dira; haiekin, klase desberdinetako objektuak era homogeneoan erabil ditzakegu (polimorfismoa).

3.1. Interfazeak

Ariketa honetan, orain arte sortu diren jokoak inplementatuko dituen interfazea sortuko da —`Jostagarria` deituko diogu—. Joko guztiek inplementatu beharreko gutxieneko eragiketa komunak bilduko ditu interfaze horrek. Horrez gain, aurrerago joko denak era generikoan erabiltzeko modua emango digu.

- `Jokoa` klasea
  - `Jokatu` metodo abstraktua ezabatuko da, baina klase honetako instantzien sorrera zuzena ekiditeko klasea abstraktu gisa mantenduko da.
- `Jostagarria` interfazea
  - `Jokatu` metodoa izango du; metodo horrek `Jokoa` klasetik kendu den metodoak egiten zuen eragiketa berbera egingo du.
  - `IzenaErakutsi` parametro gabeko metodoa gehituko da; horri esker, interfazea inplementatzen duten jokoek jokoaren izena bistaratuko dute pantailan.
  - `InfoErakutsi` parametro gabeko metodoa gehituko da; metodo horrek nola jokatu azaltzen duen mezua bistaratuko du pantailan.
- `ZenbakiaAsmatuJokoa` klasea
  - `Jostagarria` interfazea inplementatu behar du.
  - `IzenaErakutsi` metodoak pantailan honako mezu hau bistaratuko du: `Zenbaki bat asmatu`

  - `InfoErakutsi` metodoak nola jokatzen den bistaratuko du, jokalari bakoitzari ematen zaion saiakera kopurua adieraziz.

- `BikoitiaAsmatuJokoa` klasea
  - `IzenaErakutsi` metodoa hau zera bistaratzeko birdefinitzen da:
    `Zenbaki bikoitia asmatu`
  - `InfoErakutsi` metodoa birdefinitzen da.

- `BakoitiaAsmatuJokoa` klasea
  - `IzenaErakutsi` metodoa hau bistaratzeko birdefinitzen da:
    `Zenbaki bakoitia asmatu`
  - `InfoErakutsi` metodoa birdefinitzen da.

- `Aplikazioa` klasea
  - `main` metodoan aipatutako jokoen objektu bana sortuko da.
  - Horrez gain, sortu diren hiru objektuetako bakoitzaren `IzenaErakutsi`, `InfoErakutsi` eta `Jokatu` metodoei deituko die.

3.2. Arrayak

Ariketa honetan interfazeen arraya sortuko dugu, horrela interfazeen erabilgarritasuna ulertzeko. Array horri esker, hiru jokoen edozein interfazeri era generikoan deitzeko gai izango gara.

- `Aplikazioa` klasea
  - `JokoaAukeratu` metodoa
    * `Jostagarria` motako objektua itzultzen duen parametro gabeko metodo estatikoa eta publikoa.
    * Inplementaturiko hiru jokoen objektu bana sortzen du.
    * `Jostagarria` interfaze motako hiru elementuko arraya sortzen du.
    * Array hori sortutako jokoetako objektuekin betetzen du. Hemendik aurrera, edozein jokori erreferentzia egiteko, `Jostagarria` interfaze motako array horrekin baino ez dugu lan egingo.
    * Pantailan hiru jokoen izenez osatutako menu bat bistaratuko da, eta erabiltzaileari bat aukeratzeko eskatuko zaio —horretarako, 0 eta 2 arteko zenbaki bat sartu beharko du—. Sartutako balioa baliozkoa ez bada, beste bat eskatu beharko du baliozkoa izan arte.
    * Erabiltzaileak sartutako zenbakiari dagokion arrayaren elementua itzultzen du.

- `main` metodoa
  - `JokoaAukeratu` metodoari deitzen dio, erabiltzaileak aukeratutako jokoari `Jostagarria` interfaze bidez erreferentzia egiteko.
  - Joko honen `IzenaBistaratu` metodoari deitzen dio.
  - Jarraian, jokoaren `InfoBistaratu` metodoari deitzen dio.
  - Joko horren `Jokatu` metodoari deitzen dio beste partida bat hasteko.
  - Azkenik, partida bukatu ondoren, erabiltzaileari berriz jokatu nahi duen galdetuko dio. Erantzuna baiezkoa bada, aurreko pausoak errepikatuko ditu.

4. Praktika

Paketeak

Erabiltzen ari garen klase eta fitxategi kopurua hazten ari da; beraz, direktorio eta paketetan sailkatzen hasiko gara.

4.1. Paketeak

Orain arte eraturiko klaseak zenbait paketetan sailkatzen saiatuko gara, eta, horren ondoren, klase horiek edozein kode-fitxategitan erabiltzen jarraitzea posible izan dadin egin beharreko aldaketak egingo ditugu.

Sailkatzeko, pakete-egitura hau erabiliko dugu:

- `jokoak` paketea, osagai hauetaz eratua:
  - `Jokoa` klasea
  - `interfazeak` paketea, barnean hau duela:
    - `Jostagarri` interfazea
  - `zenbakiak` paketea, klase hauetaz eratua:
    - `ZenbakiaAsmatuJokoa` klasea
    - `BikoitiaAsmatuJokoa` klasea
    - `BakoitiaAsmatuJokoa` klasea
- `irakasle` paketearen barnean honako hau dago:
  - `Teklatua` klasea

Kontuan izan lehenetsitako `friendly` eremu aldatzailea duen kode bat erabilezina dela beste pakete bateko kode batentzat. Eragozpen hori ekidin eta eragiketa baimendua

izan dadin, beharrezkoa izango da `public` gisa eskaintzea. Beraz, kodea pakete desberdinetan sailkatzearen ondorioz, klase, metodo eta atributuetako eremu aldatzaileak eguneratu beharko ditugu —beharrezkoa den kasuetan—. Horrez gain, kodea pakete desberdinetan egongo denez, beharrezkoak diren `import` aginduak ere jarri beharko dira.

Eclipse erabiliz pakete bat sortzeko, proiektuaren gainean eskuineko botoiarekin klik egin eta `New/Package` hautatu beharko da. Ondoren, Eclipsek paketearen izena eskatuko du; behin izena sartua, Eclipsek paketearen izena duen direktorio bat sortuko du proiektuaren direktorio zuhaitzean. Pakete horri, gainean eskuineko botoiaz klik eginez gehituko dizkiogu nahi ditugun klaseak. Pakete horretan sortuko ditugun iturri-fitxategietan behar diren `package` deklarazioak, Eclipsek berak jarriko ditu.

4.2. Classpath

4.2.1 Kontsolatik egitea

Fitxategi guztiak, `Aplikazioa.java` izan ezik, zuen `workspace` barruko `libs` direktorio batera eraman. Horretarako, agindu hauek exekutatu beharko dituzue:

```
$ mkdir -p /home/user1/zure_izena/libs
$ cd /home/user1/zure_izena/Praktika5
$ mv * /home/user1/zure_izena/libs
$ mv /home/user1/zure_izena/libs/Ap* .
```

Hori egin ondoren programa exekutatuz gero, errore bat jasoko dugu, zeinak klaseak aurkitzen ez dituela adieraziko duen. Arazoa konpontzeko, direktorio hori `classpath`-ari gehitu beharko diogu:

```
$ java Aplikazioa
$ java -cp .:/home/user1/zure_izena/libs Aplikazioa
```

Bukatzeko, jar itzazue fitxategiak berriz jatorrizko kokapenean:

```
$ mv /home/user1/zure_izena/libs/* .
$ rm -rf /home/user1/zure_izena/libs
```

4.2.2 Eclipse erabiliz egitea

Lehendabizi, irakasle-paketeari dagokion `JAR` fitxategi bat sortuko dugu. `JAR` fitxategia konprimituriko `*.class` fitxategi multzo bat baino ez da.

1. Eskuineko botoiaz klik egingo dugu paketean, eta `Export...` hautatuko dugu.

2. `Java/JAR file` aukeratu eta `Siguiente` botoia sakatuko dugu.

3. Ondoren, paketean sartu nahi ditugun fitxategiak aukeratuko ditugu —kasu honetan, `Teklatua.java`—.

4. Aurreko pausoko pantaila berean jarraituz, sortu nahi dugun fitxategiaren izena `irakasle.jar` dela jarriko dugu, eta `terminar` botoia sakatuko dugu.

Behin hori eginda fitxategi-arakatzailea irekitzen badugu, ikusiko dugu gure `workspace`-aren barruan `irakasle.jar` izeneko fitxategi berri bat sortu dela.

Ondoren, `irakasle` paketea gure proiektu-zuhaitzetik ezabatuko dugu, haren gainean eskuineko botoiarekin klik egin eta `Delete` hautatuz. Ezabatzean —`irakasle` paketea ez aurkitzearen ondorioz— Eclipsek gure kodean erroreak bistaratuko ditu.

Erroreak konpontzeko, sortu berria dugun `JAR` fitxategia gehituko diogu gure `classpath`-ari. Horretarako, urrats hauek egin beharko ditugu:

1. Eskuineko botoia erabiliz, proiektuaren gainean klik egin, eta dituen aukeretatik `Build Path/Add External Archives...` hautatuko dugu.

2. `irakasle.jar` fitxategia aukeratu, eta `aceptar` sakatuko dugu.

`classpath` berriarekin beharrezkoak diren pakete guztiak aurkitu ahalko ditu; egiazta ezazue Eclipsek aurreko exekuzioan bistaratutako erroreak desagertzen direla.

5. Praktika

APIa

Praktika honen helburua zera da: ikasleak kanpoko pakete baten dokumentazioa irakurri eta erabiltzeko gai izatea. Horretarako, Javako API[1]aren `Random`, `String` eta `Vector` klaseak erabiliko dira.

5.1. Random klasea

Javako APIaren dokumentazioa begiratuz, `java.util` paketeko `Random` klasea erabili beharko da. Klase hori, asmatu beharreko zenbakia programatzaileak jarritakoa izan beharrean ausazkoa izan dadin erabiliko dugu. Horretarako:

- `ZenbakiaAsmatuJokoa` klasea
  - Asmatu beharreko zenbakia ez du jada parametrotzat hartuko eraikitzaileak.
  - Datu kide moduan, gehitu ezazu ausazko zenbakiak sortzeko erabiliko dugun `Random` motako objektu bat. Objektu hori sortzeko orduan, komeni da hasierako hazi bat pasatzea; era horretan, sahiestu egingo da beti zenbaki-sekuentzia bera sortzea. Hazi modura, `java.util` paketearen `Date` klasea erabil daiteke.
  - `PartidaBerrasi` metodoa birdefinitu, orain arte `Jokoa` klasean zuen kodea exekutatzeaz gainera asmatu beharreko zenbakiari ausazko balio bat eman diezaion.
- `BikoitiaAsmatuJokoa` klasea
  - `PartidaBerrasi` metodoa birdefinitu, `ZenbakiaAsmatuJokoa` klasean zuen kodea exekutatzeaz gainera, kode horrek sortutako ausazko zenbakia 0 eta 10 artean dagoen zenbaki bikoiti batera egokitu dezan.

[1] `http://java.sun.com/javase/reference/api.jsp`

- `BakoitiaAsmatuJokoa` klasea
  - `PartidaBerrasi` metodoa birdefinitu, `ZenbakiaAsmatuJokoa` klasean zuen kodea exekutatzeaz gainera, kode horrek sortutako ausazko zenbakia 0 eta 10 artean dagoen zenbaki bakoiti batera egokitu dezan.

5.2. String klasea

`String` klasea erabiliz joko bat —oraingoan, zenbakietan oinarritu gabe— inplementatuko dugu: `UrkatuJokoa`

- `UrkatuJokoa` klasea
  - `jokoak.hizkiak` paketean egongo da.
  - `Jokoa` klasetik eratorria izango da.
  - Eraikitzailearen lehen parametroa bizi kopurua izango da; eta bigarrena, asmatu beharreko karaktere-katea.
  - `Jostagarria` interfazea inplementatuko du.
  - `Jokatu` metodoa inplementatzeko, honako pauso hauek egitea gomendatzen da:
    * Oinarrizko klasearen `PartidaBerrasi` metodoari deitu.
    * Asmatu beharreko katearen tamaina berdina duen katea sortu, eta '-' karakterea jarri posizio guztietan.
    * Erabiltzaileari '-' karaktereak dituen katea erakutsi.
    * Erabiltzaileari karaktere bat eskatu, eta asmatu beharreko katean dagoen baieztatu.
    * Karaktere hori katean badago, dagoen posizioetan, '-' karakterea asmatutako karakterearekin ordeztu. Katearen balio berri hori asmatu beharreko katearen balioarekin konparatu, eta, berdinak badira, erabiltzaileari adieraziko zaio, eta partida bukatu beharko da.
    * Karakterea katean ez badago, `BiziaKendu` metodoari deitu eta partida bukatu den ala ez ziurtatu. Partida bukatu ez bada, '-' karaktereak dituen katea berriro bistaratu eta prozesua errepikatu.

- `Aplikazioa` klasea
  - `JokoaAukeratu` metodoari sortu berri den jokoa gehitu.

5.3. Vector klasea

`java.util` paketearen `Vector` klasearekin praktikatzeko, `Aplikazioa` klaseko interfazeen arraya bektore batez ordeztuko dugu.

- `Aplikazioa` klasea
  - `InfoBektorea` metodoa gehitu. Metodo horrek parametro moduan bektore bat hartu, eta haren tamaina eta edukiera pantailan bistaratuko ditu.
  - `JokoaAukeratu` metodoan bektore bat sortu, hiru edukiera eta jauzietan bi elementuko gehikuntza dituena. Ondoren, `InfoBektorea` metodoari deitu.
  - Bektore horri zenbakien hiru jokoak gehitu, eta `InfoBektorea` metodoari berriz deitu.
  - Jarraian, bektoreari urkatuaren jokoari dagokion objektua gehitu, eta berriro `InfoBektorea` metodoari deitu.
  - Funtzioak interfazeen arrayarekin lan egin beharrean bektorearekin lan egin dezan beharrezkoak diren gainerako aldaketak egin.

6. Praktika

Salbuespenak

Azken praktika honetan, erabiltzaileak sartzen dituen datuak balioztatzeko, salbuespenen erabilerari buruzko gogoeta egingo da.

6.1. 1. ariketa

`Teklatua` klaseko `OsoaIrakurri` metodoak 0 balioa itzultzen du sarturikoa zenbaki bat izan ez bada. Metodo hori aldatu, orain `NumberFormatException` motako salbuespen bat jasoz gero erabiltzaileari zenbaki zuzen bat sartu behar duela adieraz diezaion eta sarrera berriz irakur dadin. Jasotako salbuespena beste mota batekoa baldin bada, 0 balioa itzultzen jarraituko du.

6.2. 2. ariketa

Klase-eraikitzaileak ezin du itzulera-koderik itzuli eragiketa zuzena izan den ala ez adierazteko. Hori ekiditeko, bada irtenbide bat: baldin akatsik badago, salbuespen bat sortzea.

- `JokoaException` klasea
  - `jokoak.salbuespenak` paketean egongo da.
  - `Exception` klasetik eratorritakoa izan beharko du.
  - Haren eraikitzaileak salbuespenaren zergatia deskribatzen duen karaktere-kate bat hartuko du parametrotzat.
- `UrkatuaJokoa` klasea

  - Birmoldatu eraikitzailea, baiezta dezan asmatu beharreko hitzaren karaktereak ez direla zenbaki; horretarako, `Character` klaseko metodoak erabil daitezke. Zenbakirik balego, `JokoaException` motako salbuespen bat jaurtiko du hori adierazteko.

- `Aplikazioa` klasea

  - `JokoaHautatu` metodoak ez du salbuespena jasoko, gorantz pasatu baizik.
  - `main` metodoak edozein salbuespen jaso beharko du, eta, bukatu aurretik, akatsaren zergatia adierazi.
  - Salbuespenen bat jaso bada, zein jaso ez bada, `Programaren amaiera` mezua bistaratuz bukatu beharko da.

Eginiko aldaketak frogatzeko, programa konpilatu eta exekutatu kasu bietan; asmatu beharreko hitzak zenbaki bat duenean eta zenbakirik ez duenean.