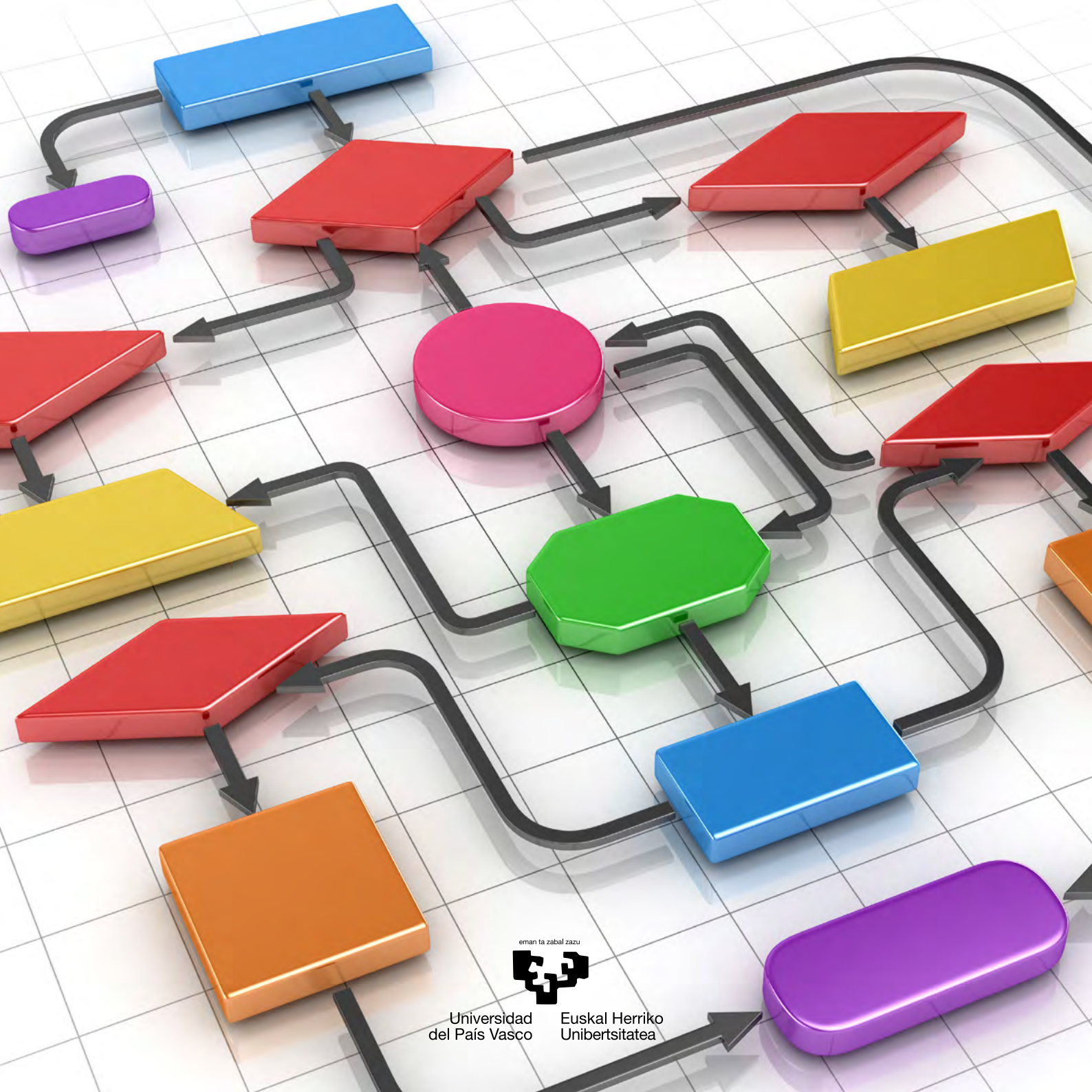
Oinarrizko Programazioa

Iker Azpeitia Lakuntza
Jesús Ibáñez Martínez-Conde



eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

CIP. Unibertsitateko Biblioteka

Azpeitia Lakuntza, Iker

Oinarrizko programazioa [Recurso electrónico] / Iker Azpeitia Lakuntza, Jesús Ibáñez Martínez-Conde. – Datos. – Bilbao : Universidad del País Vasco / Euskal Herriko Unibertsitatea, Argitalpen Zerbitzua = Servicio Editorial, [2020].

1 recurso en línea : PDF (85 p.)

Modo de acceso: World Wide Web.

ISBN: 978-84-1319-253-6.

1. Programas y sistemas de programación. 2. Lenguajes de programación. I. Ibañez Martínez-Conde, Jesús, coaut.

(0.034)681.3.06

UPV/EHUko Euskara Zerbitzuak sustatua eta zuzendua, Euskarazko ikasmaterialgintza sustatzeko deialdiren bitartez.

ISBN: 978-84-1319-253-6

Aurkibidea

1. gaia

Sarrera

Eskuliburu honen helburua informatikaren oinarriak erakustea da, programazioari dagozkion funtsak aditzera eman arte. Gipuzkoako Ingeniaritza Eskolako (Eibarko atala) Informatika irakasgaian erabiltzeko sortu da. Energia Berriztagarrien ingeniariengan pentsatuz idatzi den arren, testuan zehar erabiltzen diren adibideak orokorrak dira. Era honetan, edozein gradutako ikasleek ere erabil dezakete. Are gehiago, ez da aldez aurretik jakintza aurreikusten aipatutako kontzeptuak ulertzeko; beraz, interesa duen pertsona orok erraz erabil dezake testu hau.

Pertsonalki, informatika arloko irakasle bezala, eskuliburu honekin genituen helburu xumeak lortu ditugula pentsatzen dugu:

— Helburu nagusia Eibarko Informatika ikasgaiko irakaskuntza hobetzea zen. Informatika ikasgaian programazioaren oinarrizko kontzeptuak azaldu eta lantzeko ikasliburu bat edukitzearekin aurrerapen txiki bat lortu dugula uste dugu. Liburuak klaseko apunteak baino landuagoak egoten dira, urteetan iraun dezaten. Gainera, kurtso hasieran ikasleentzako eskuragarri egongo da, ikaskuntza autonomoa bultzatuz.
— Beste helburua euskarazko ikasmaterialen sorrera bultzatzea izan da. Euskaraz programazio-lengoaia zehatzetan programatzen irakasten dituzten testuak badaude ere (ikus erreferentzien atala), programazioaren metodologia erakusten duten aukerak urriagoak dira. Eskuliburu honekin gure ekarpen txikia egin nahi izan dugu.

Espero dugu irakurlearen gustukoa izatea. Edozein iradokizun edo akats-zuzenketa ongietorria izango da. Gure posta elektronikora bidalita, irakurriko dugu.

Zer da informatika?

Berez *informatika* hitza «informazioa» eta «automatikoa» terminoak elkartzetik eratorri da. Hain zuzen ere, diziplina honen lorpen guztiak datuak automatikoki prozesatzeko gailu baten asmakuntza eta garapenean neurtzen dira: ordenagailua. Baina informatikaren existentziaren baldintzak ezartzeko konputagailua instrumentala bada, aipagarria da informatikaren legeak bide independente eta paraleloan garatu izan direla. Egia esan, informazioa prozesatzearen ezaugarriak eta murriztapen teoriko nagusiak ordenagailua agertu baino zenbait urte lehenago ezarri ziren. Hasieratik, beraz, informatikaren oinarriak Independentzia Legeetan sustatzen direla jakin izan dugu:

— Informazioa burutu, gauzatu eta manipulatzeko euskarri fisikoak *(hardware)* ez dauka eraginik informatikaren prozesamenduaren ezaugarrietan. Egun ezagutzen ditugun konputagailuak zenbait arbaso mekaniko eta elektromekaniko izan dituzte, eta teknologia elektronikoaren aukera izan da abiadura eta kostuaren inguruko arduraren ondorioa.
— Hardware fisikoaren gainean informazioaren propietate immaterialak erabiltzen dituen geruza logikoa *(software)* dago. Bertan, izate eta prozesu abstraktuen bidez mamitzen dira informazioa eta bere tratamendua, sustapen fisikoaren baldintzak erabat baztertuz. Izate eta prozesu horien deskribapen eragilea da programazioa.

Informatika:	«Informazioaren tratamendu automatikoaren zientzia. [ ] Zientzia honen funtsa datuak bildu, sarrera-unitateen bidez ordenadorera sartu, bertako unitate zentralean prozesatu ondoren informazio erabilgarria sortu eta irteera-unitateen bidez ateratzean datza. Prozesu honetan bi kontzeptu nagusi bereiz daitezke: hardwarea, hots, prozesamendu automatikoan erabiltzen diren tresnen multzoa, eta *software*a ordenadoreak funtzionatzeko eta problema konkretuak ebazteko behar diren programen multzoa. [...]». *(Harluxet)*

Nolabait, robotak gihar-indarra bideratzen duen bezala, konputagailuak burmuin-indarra bideratzen du, bi kasuetan mekanika eta elektronika gizakiaren helburuak lortze aldera jarriz. Baina robotaren arrakasta bere ezaugarri fisikoetan datza. Biltegia antolatzeko robotak higikorra izan behar du, korridoreetan maniobrak egiteko dimentsio egokiak eduki behar ditu, fardelak altxatzeko adina indartsua izan behar du... Horietako ezaugarri bakar bat asetzen ez badu, ezin izango du betebeharra burutu. Konputagailuak, ordea, behar duen osagai bakarra bere nolakotasuna da: *software*/*hardware* dualtasuna gauzatu. Programagarria den geruza logikoa, eta programa gauza dezakeen geruza materiala.

Informatikaren lehengaia izanik, informazioaren izaerari buruzko hainbat zehaztapen egiteko ordua heldu da. Materialtasunetik duen independentzia aipatu dugu; baina, orduan, zertan oinarritzen da? Informazioaren muina kontrakotasunean datza. Dauden posibilitateen artean bat hautatzen denean, hori markatu egiten da, baina, era berean, baztertu diren aukera guztiak ere bai. Pertsona bat A odol-taldekoa dela ikasten badugu, beste hiru taldetakoa ez dela ere jakingo dugu. Horrez gain, Y-kromosomako A haplotaldekoa ere badela esaten badigute, *beste hamaseietan* ez dagoela ere ezagutuko dugu. Intuitiboki bigarren izaerak informazio gehiago ematen digu lehenengoak baino. Informaziorako bereizgarriak diren aukerak behar ditugu, eta horiek adierazteko *ikurrak* premiazkoak dira. Alfabetoak, zenbaki-sistemak, ikonoak, kolore-kodeak, ganadua xakitzeko markak edo emojiak aukeren artean bereizteko ikur multzoak dira. Baina kontrakotasun-sistema guztiak bi posibilitateen arteko aukera anitzen gainean eraiki daitezke. Bi ikur bereizgarri nahikoak dira hautaketa minimo hori egiteko, logikan adierazten den oinarrizko aukeran bezala: *egiazkoa* ala *faltsua*. Motibo historikoak direla medio, informatikan bereizketa bitar hori **0** eta **1** ikurren artean planteatzen da.

Aukerak kodetzeko ikurren bidez informazioaren alderdi teoriko guztiak deskriba eta azter daitezke, baina hori ez da liburu honen helburua. Gehiago interesatzen zaigu ikurren eta errealitatearen arteko erlazioa nola ezartzen den, hau da, nola definitzen diren informazioaren propietate semantikoak. **0** ikurra datua da adierazpide bereizgarria den heinean, baina berez ez dauka erlazio berezirik mundu errealeko entitateekin. Ordenagailuak ez du erlazio horretaz ezer jakin behar datu horrekin lan egiteko, baina gizakioi zaila egiten zaigu datuak manipulatzea ez baditugu in-

terpretatzen. Hots, *informazioaren esanahia* gizakiok sortzen dugu *datuen interpretazio*an. Adibidez, **0** datua ikasle baten nota baldin bada, informazio oso garrantzitsu bilaka daiteke ikaslearentzat.

Interpretatzeko ahalmenik gabe ordenagailuek datuak prozesatzen dituztenez, gizaki eta ordenagailu binomio horri *sistema informatiko* deritzo. Hiru maila ditugu sistema honetan:

1. Gizakiok egiten dugun **problemaren definizioa eta interpretazioa** (problemaren modeloa).
2. Ordenagailuak jarraitzeko **prozesuaren programa** (problema ebazteko softwarea).
3. Elkarlanean dabiltzan **kalkulurako osagai elektronikoak** (kalkulu eraginkorrak egiteko hardwarea).

Software eta hardwarea funtzioak ondo banatu eta koordinatzeko mekanika automatikoki gauzatzen bada ere, gizakiaren eta ordenagailuaren arteko komunikazioa ezin da aurretik programatu, errazteko gailuak etengabe hobetzen badira ere. Gizakiak problemaren datuak ditu, ordenagailuari helarazi behar dizkio, honek kalkuluak egin ditzan, azkenik emaitza-datuak bueltan jasotzeko. Beraz, *datuen prozesaketan* oso garrantzizkoa da pertsonaren eta makinaren arteko komunikazio alderdi horiek islatu eta bereiztea, hiru fase azpimarratuz:

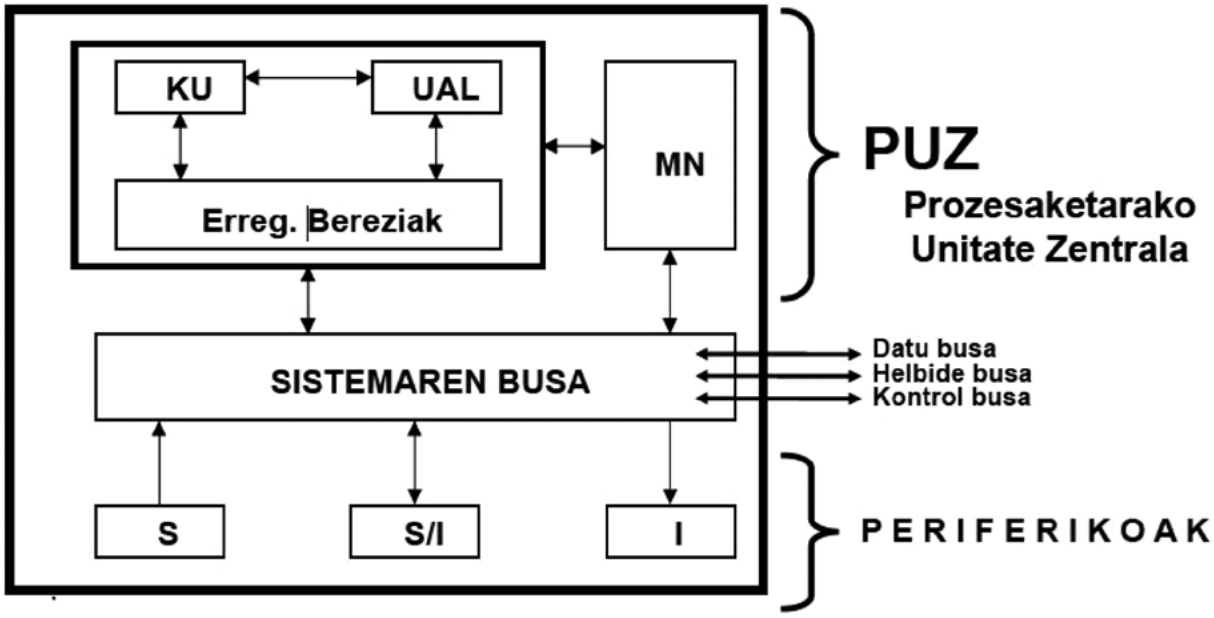
Sarrera ⇒ Prozesaketa ⇒ Irteera

Sistema informatikoaren mailak eta prozesaketaren faseak elkarrekin jartzen baditugu, honako taula hau izango genuke. Maila logikoan, gizakiok problema zehazten dugu, hasierako eta emaitzen informazioa erlazionatuz. Software mailan, programak datuak jasotzen ditu, aginduak jarraituz tarteko emaitzak kalkulatzen ditu, eta azken emaitza bueltatzen du. Hardware mailan, gailu fisikoek sarrera-datuak ordenagailura barneratzen dituzte, irteera-gailuek emaitzak erabiltzaileari aurkezten dizkiote, eta Prozesaketarako Unitate Zentralak (PUZ) kalkuluak egiten ditu.

	Datu-sarrera	**Datu-prozesaketa**	**Datu-irteera**
Problemaren modeloa	Erabili beharreko datuak zehaztu	Problema ebazteko metodoa zehaztu	Lortu beharreko emaitzak zehaztu
Problema ebazteko programa (SW)	Datuak jaso sarrera- aginduen bitartez	Prozesaketa-aginduak jarraitu	Emaitzak erakutsi irteera-aginduen bitartez
Kalkulurako gailuak (HW)	Sarrerako gailuek datuak kodetzen dituzte	Prozesaketarako Unitate Zentrala (kalkuluak) + Memoria (datuak)	Irteerako gailuek datuak aurkezten dituzte

Problema bat ebazten saiatzen garenean, ahalik eta soluzio orokorrena aurkitu nahi dugu. Problemaren instantziek soluzioaren aplikazio konkretua eskatuko dute, eta, horretarako, sarrera-datuak eskatzen dira, problema zehatz bat definitzeko. Taulak dioen bezala, software mailan **datu-sarrerako aginduak** izango ditugu hasierako balio horiek eskatzeko. Ondoren, **prozesaketarako aginduek** datu horietatik emaitza-datuak kalkulatuko dituzte. Azkenik, **irteera-aginduek** emaitzak erakutsiko dituzte edo fitxategietan gordeko.

Hardware mailako konputagailuen oinarrizko arkitektura adierazten da honako grafiko honetan:



Laburki, lehen adierazi ditugun datu-prozesaketarako hainbat gailu bereizgarri dauzkagu: Prozesaketarako Unitate Zentrala eta sarrera-irteerako gailu periferikoak. Horien arteko datu-trukaketa egiteko, sistemaren busa dugu. Hau da, hariak elektronikoki datuak, memoria-helbideak eta aginduak bideratzeko.

Zehazki, **sarrerako gailuek** (teklatua, sagua, mikrofonoa, eskanerra...) datuak ingurune fisikotik hartzen dituzte, eta informazio digital bihurtzen dituzte. Lehenago aipatu dugun bezala, horrek esan nahi du jasotako informazio hori ikur-segida bihurtu behar dela. Kasu batzuetan, digitalizatzeko lana zuzen samarra da. Teklatuaren kasuan erabiltzaileak informazio sinbolikoa sartzen du, eta tekla sakatzeak sorrarazten duen seinale elektrikoa erraz eraldatzen da behar den ikurretan (letra, digitua...). Aldiz, beste sarrera-gailu batzuen lana ez da hain nabarmena. Saguak bere mugimendu horizontala detektatu behar du sentsore mekaniko ala optikoa erabiliz, mugimendu hori espazio-koordenatu digitaletan adierazi behar du (askotan posizioa, abiadura eta azelerazioa ere beharrezkoak dira), eta lortutako koordenatuak konputagailuak maneiatzen duen koordenatu-sistema abstraktuan interpretatu behar dira (adibidez, aktiboa den leihoarena). Sarrerako gailuak **S** letraz adierazten dira konputagailuen arkitekturaren deskribapenean.

Datu digitalak **memoria nagusira** (**MN**) bideratzen dira **sistemaren busaren** bitartez. Memoria nagusian programak erabiliko dituen laneko datuak gordetzen dira. Aipagarria da egikaritzen d(ir)en programa(k) ere memoria nagusian egon behar d(ir)ela, informazioa prozesatzeko ezinbesteko datu digitalak izanik.

Prozesaketarako Unitate Zentrala (**PUZ**) da informazio aktiboa (aginduak) eta pasiboa (datuak) bereizten dituena. Horretarako, programaren aginduak banan-banan irakurri eta egikaritzen ditu, eta agindu horiei esker jakingo du zein diren (edo hobeto, non dauden) maneiatu behar dituen datuak. Bere osagai nagusiak honako hauek dira:

— **Kontrol Unitateak** (**KU**) aginduak dioen eragiketa eta eragileak identifikatzen ditu. Funtzio zehatzak dituzten erregistroetan antolatuta dago, eta eragileak memoria nagusitik bertara pasatzen dira prozesatu behar direnean.
— **Unitate Aritmetiko Logikoa** (**UAL**) eragiketak egikaritzen espezializaturik dago. Beraz emaitza berriak lortzen ditu Kontrol Unitateak bere enkarguak bete ditzan.

Batzuetan, tarteko emaitzak kalkulatu behar dira, eta berriro ere memoria nagusira bidaltzen dira geroko beste eragiketetan erabiltzeko. Horrela, programak dioena eginez, azken emaitzak lortu arte.

Era berean, **irteerako gailuek** erabiltzaileari azken emaitzak erakusten dizkiote (pantaila, inprimagailua, bozgorailua...). Irteerako gailuak **I** letraz adierazten dira konputagailuen arkitekturaren deskribapenean. Aurreko kasuan ez bezala, egun ez dago lan erraza duen irteerako gailurik. Teknologia heldu aurretik, pantailak zein inprimagailuak karakterekoak ziren, eta orduan esan zitekeen nahiko mekanikoa zela emaitzak aurkezteko lana. Irteerako datuen ikurrak gizakiak ikus dezakeen errepresentazioaz baino ez ziren adierazi behar, karaktere bakoitzerako tinta- edo argi-puntu gutxi nahikoa izanik. Gaur egun jasotzen ditugun datuen aurkezpenetan, testua, grafikoa, soinua, animazioa, bideoa eta beste hainbat formatu eta aukera integratzen dira era naturalean. Baina «naturaltasun» horrek atzeko planoko lana erraldoia eta oso konplikatua du.

Kasu batzuetan, ordenagailuak hartzen edo ematen dituen datuak beste ekipo batean ala beste momentu batean sortuak/erabiltzekoak dira. Kasu horietan **sarrerako/irteerako gailuei** (**S/I**) buruz hitz egiten dugu, eta S/I etiketa daukate (disko gogorra, DVD grabatzailea, USB unitatea...). Gailu horietan kezka nagusia ez da gizakiarekin komunikatzea, beste konputagailuekin baizik. Beraz, zailtasun nagusia munduko ekipoen artean dagoen aniztasunean datza: teknologia ezberdinak, manufaktura ezberdinak, software ezberdinak, kultura ezberdinak, merkataritza-interes ezberdinak... Horrek guztiak esan nahi du gailu horien diseinuetarako estandarren bilaketa aurrebaldintza dela.

Zorionez, aipatu ditugun prozesuak, gardenak dira erabiltzailearentzat. Jakin badakigu datuak sartzeko eta erauzteko erabiltzen diren prozesu fisikoak ez direla ezagutu behar, sarrera/irteera-gailuen erabilpena menperatzea aski baita. Era berean, konputagailua programatzeko periferikoak eta software berezia dira tresna nagusiak, egindako programak martxan jartzeko eta exekutatzeko zehazkizunak automatizaturik baitaude.

Zertarako eta noiz erabili Informatika?

Ordenagailuak programan adierazitako aginduak ulertu eta betetzen ditu gailu elektronikoen bitartez; hau da, hardwareak softwareak adierazten duena egiten du. Ordenagailuek prozesaketa automatikoa era egokian egiten dute ezaugarri hauek betetzen dituztelako:

— **Fidagarriak** dira. Ez dute errorerik sortzen. Ez, behintzat, beren kabuz. Programek akatsak baldin badituzte, programatzailearen erruz izaten da. Gainera, hardwarea (RAM memoria, CD irakurgailua...) matxuratzen bada, alda daiteke.
— **Azkarrak** dira. Kalkulu ugari denbora gutxian. Ordenagailuen kalkulu-potentzia FLOP (Floating Point Operations per Second) neurrian adierazten da. Hau da, segundoko zenbat eragiketa sinple egikari ditzaketen. Gaur egungo ordenagailuen kalkulu-potentzia Te-

raFLOPetan adierazten da, 10 ber 9 eragiketa segundoko, alegia. Gerta daiteke programa batek eragiketa amaiezinak egitea, baina, berriro ere programatzailearen errua izango litzateke.

— **Zehatzak** dira. Zenbakiak nahi adina zehaztasunarekin adieraz ditzakegu. Adibidez, M programazio-lengoaian erabil dezakegun zenbaki positibo handiena 1.7977e + 308 da, eta txikiena 2.2251e – 308.
— **Errentagarriak** dira. Teknologiaren potentzia urtetik urtera handitzen doa; kostua, aldiz, mantentzen ala gutxitzen.
— **Moldagarriak** dira. Alde batetik, ordenagailuek gailu berriak txertatzea ahalbidetzen dute, potentzia zein egin beharreko berriak onartzeko. Bestetik, ordenagailuak helburu orokorrekoak dira; beraz, egin beharreko zehatz batera bideratzeko programa zehatza egikaritu behar dute. Horren abantaila da ordenagailu bakarrarekin egin beharreko asko gauza ditzakegula programa desberdinen bitartez.

Helburua gizakiok dugun problema bat ebaztea da. Nabarmentzekoa da honen guztiaren muina problema ebazteko metodoa dela. Ordenagailuek ez dakite nola ebatzi problema bat, gizakiek adierazi behar diete nola egin. Hau da, **programatzaileak** (programa sortzen duen gizakiak) esan behar dio ordenagailuari zein prozesaketa egin behar duen sarrerako datuetatik emaitzak lortzeko. Hori **programa** baten bitartez adierazten da.

Programa:	«Ordenadore bati programazio-lengoaia batean idatzita ematen zaion agindu-segida, lan jakin bat burutzeko behar diren oinarrizko eragiketak berak egikaritzeko moduan zehatz azaltzen dituena». *(Harluxet)*

Baina, programatzeak denbora eskatzen du. Horregatik, programatzaileak kontuan hartu beharko du programa bat sortzea komeni den ala ez. Denbora-tarte txiki batean gure kabuz (arkatzez eta paperez edo bestelako tresnen bitartez) problema bat ebatz badezakegu, ez dugu denborarik galduko programa bat idazten. Beraz, honelako ezaugarriak dituzten problemetan programa bat sortzea komeni daiteke:

— **Datu asko** ditugunean edo ohiko ataza errepikakorrak egin behar ditugunean. Gizakiak aspertu egiten gara, eta horrelakoetan kalkulu okerrak egiten ditugu; ordenagailuek inoiz ez.
— **Kalkuluak konplexuak** direnean. Ordenagailuek azkartasunez, zehaztasunez eta errorerik gabe egingo dituzte.
— **Datuak partekatuak** direnean, bikoizketak ekiditeko. Internet bitartez, ordenagailuen arteko komunikazioa azkarra da.

Ikus dezagun! Pentsatu adibide sinple honetan: bi balioaren bikoizketa. Berez, eragiketa erraza da, eta ez genuke horretarako programa bat idatziko; buruz egin dezakegu. Saia gaitezen! Adierazi honako ariketa hauen emaitza:

2 + 2 = ?

hamar aldiz 2 + ... + 2 = ?

ehun aldiz 2 + ... + 2 = ?

Seguru asko buruz errorerik gabe egin dugu, segundo gutxitan. Baina ez genuke ordenagailu-programa bat sortuko problema horiek ebazteko. Hurrengo adibide honetan eragiketa aldatuko dugu: bi balioaren bikoizketa. Adierazi honako ariketa hauen emaitza:

*2 * 2 = ?*

*hamar aldiz 2 * ... * 2 = ?*

*ehun aldiz 2 * ... * 2 = ?*

Eragiketa hau pixka bat konplexuagoa da, eta askotan egin behar badugu ez dugu asmatzen. Orduan, agian, programa baten beharra izango genuke.

Problema mota desberdinak daude. Batzuetan, aurreko adibidean kasu, helburua eragiketa matematikoen emaitza lortzea da. Beste batzuetan, denboran zehar sistema konplexuen portaera aurreikusi nahi da, simulazioetan kasu. Simulazioetan, aurreko segundoko emaitza ondorengo segundoko datu-sarrera da, emaitza berri bat lortzeko, eta horrela hainbat segundotan. Horrelakoetan (datu eta kalkulu asko, azkartasun beharra, dinamikoa...), egokia izaten da programa informatiko bat.

Programen **berrerabilpena** informatikaren abantaila handi bat da. Programa bat behin egin eta askotan erabiltzeak errentagarritasuna areagotzen du, eta egon daitezkeen erroreak azalarazten ditu (ondoren erroreak zuzentzeko). Berrerabilpena ahalbidetzeko programek orokorrak izan behar dute; hau da, problema baten kasu gehienak ebatzi behar dituzte. Horretarako, programa parametrizatu egin beharko da. Uler dezagun adibide matematiko batekin. Demagun karratua ebazten duen programa bat dugula; X ber 2 motako problemak ebazten ditu. Oinarria edozein izanda, bere karratua lortzen dugu. Orokorragoa izango balitz, adibidez berretzailea parametrizatuz (X ber Y), karratuak kalkulatu ahal izango genituzke (X ber 2), eta baita erro karratuak ere (X ber ½).

Zein dira programazioren urratsak?

Programak sortzea nahiko artistikoa da. Esan bezala, ordenagailuak oso egokiak dira problema konplexuak zuzen eta azkar ebazteko. Baina, horretarako, ordenagailuek programan adierazitako prozedura jarraitu behar dute. **Analista programatzailea**[1] da, problema ulertuz, programa diseinatzeaz eta idazteaz arduratzen dena. Beraz, programatzaileak programa erabiliko dutenen (hots, **erabiltzaileak**) problema ulertu eta ebazpen-prozedura bat diseinatu beharko du.

Analista programatzailea:	«Ordenagailuz tratatu ahal izateko, problemak analizatu eta gero hortik sortutako programazio-lanak egiten dituen pertsona». *(Euskalterm)*

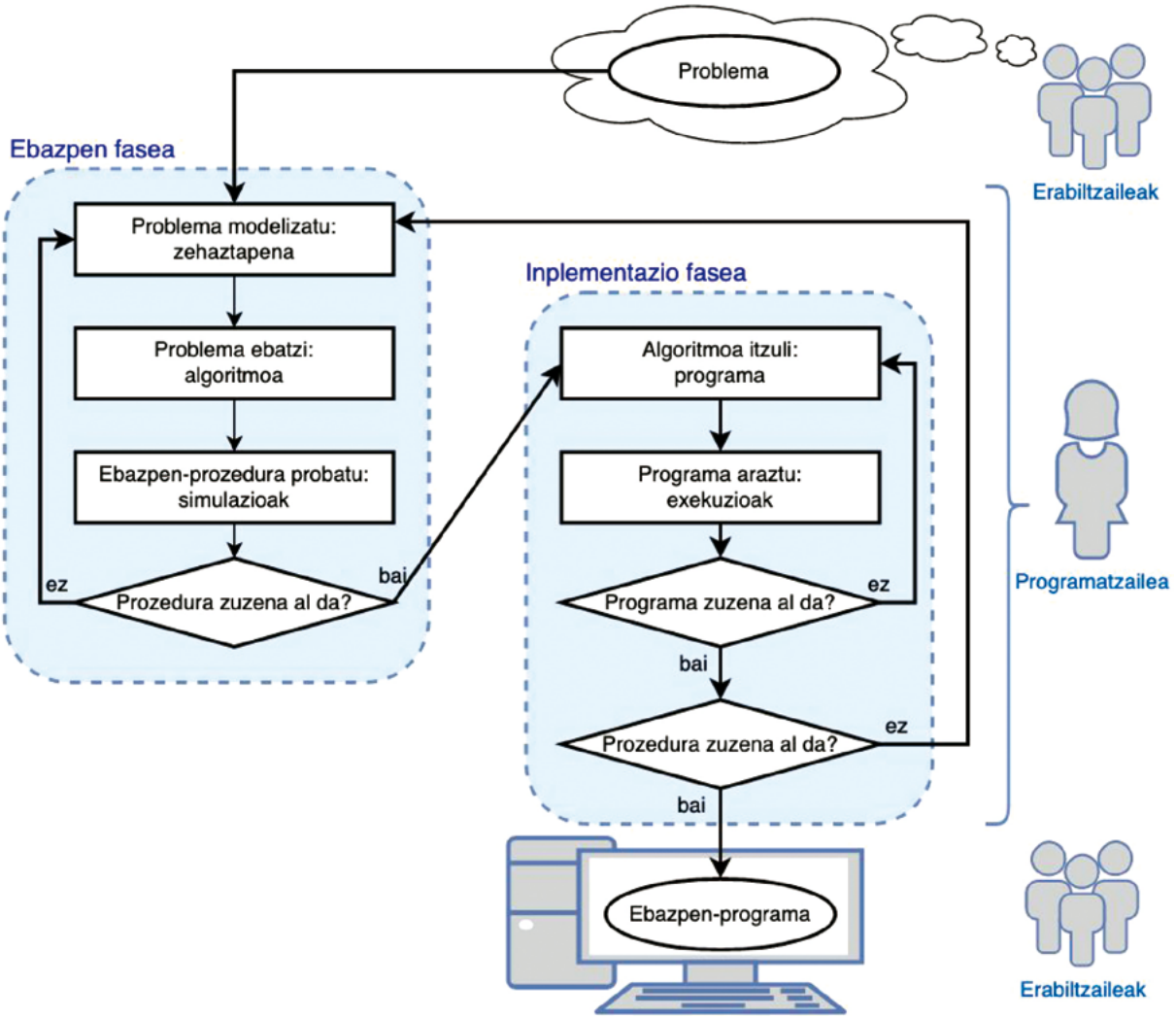
Erabiltzailea:	«Ordenadore edo sistema informatiko bat erabiltzen duen pertsona, informatikan aditua izatea behar ez duena». *(Harluxet)*

[1] Eskuliburuan zehar, «*analista programatzaile*» hitza «*programatzaile*» bezala laburtuko dugu.

Programatzerako orduan, bi fase daude: ebazpen fasea eta inplementazio fasea (ikus **1. irudia**). Ebazpen fasean, problema ulertu, definitu eta algoritmo baten bitartez ebazten da. Lehenengoz, problemaren modeloa egiten da sarrerako (hasierako informazioa) eta irteerako (emaitzak) datuak zehaztuz (**zehaztapena**). Ondoren, irteerako emaitzak lortzeko **algoritmo** baten bitartez adieraziko da sarrerako datuak nola prozesatu behar diren. Algoritmoak eskuz eginiko programen diseinuak dira. Ondorioz, algoritmoak egokiak diren probatzeko kasu berezien eta kasu orokorren **exekuzio-simulazioak** egingo dira. Emaitzak zuzenak badira, orduan algoritmoa **programazio-lengoaiara** itzuliko da.

Algoritmoa:	«Problema baten ebazpenerako eman behar diren urratsen deskribapen formala. Programazio-lengoaia baten bidez, algoritmoa ordenadore batek egikari dezakeen programa bihur daiteke». *(Harluxet)*

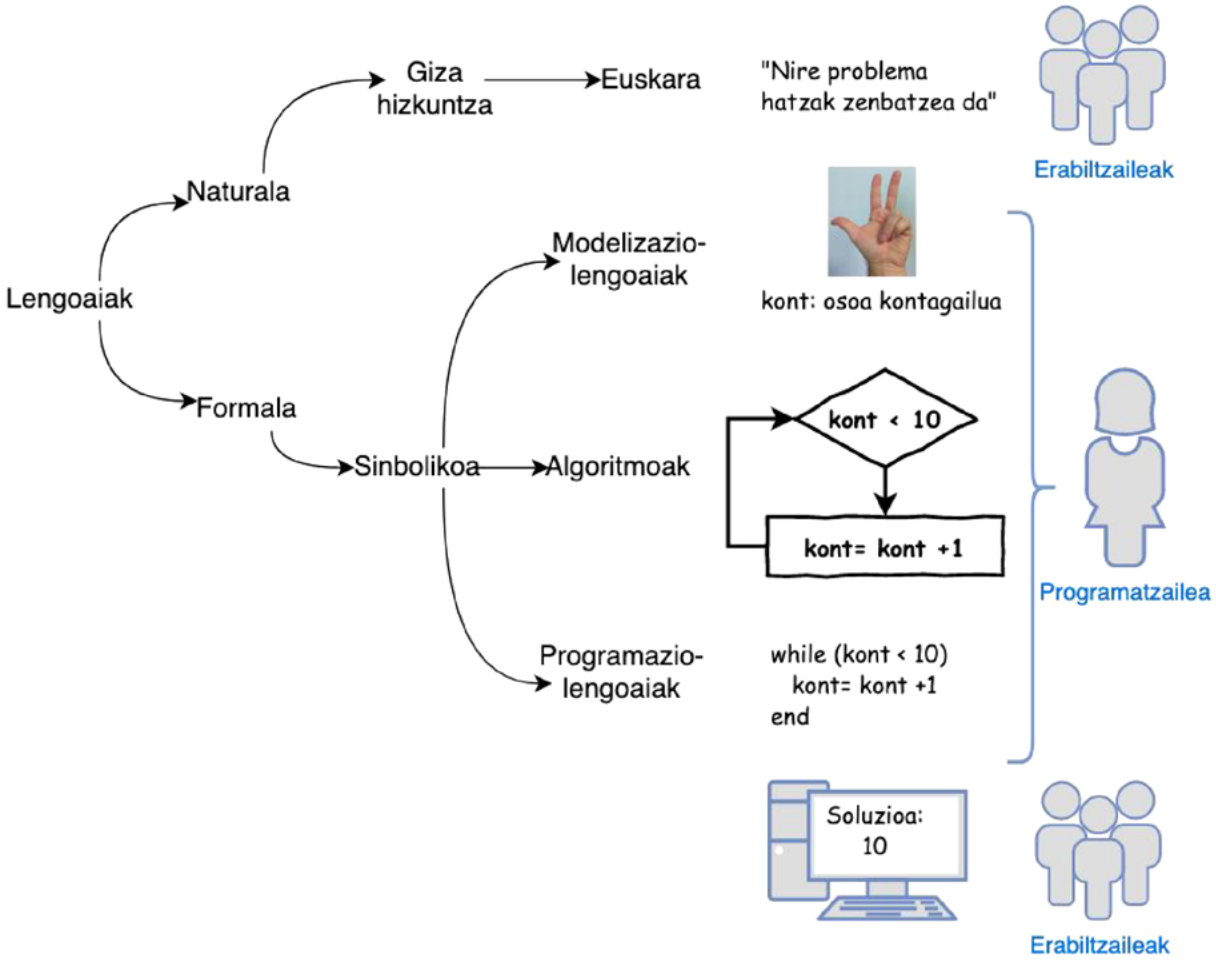
Programazio-lengoaia:	«Lengoaia artifiziala, bereziki eta espresuki ordenagailu- programak adierazteko sortuak». *(Euskalterm) (Harluxet)*



1. irudia. Ebazpen eta inplementazio faseak

Inplementazio fasean programazio-lengoaia bat erabiliz idazten da programa. Algoritmoak dioena programazio-lengoaia zehatz batera itzultzen da; izan ere, ebazpen fasean jada problema ebatzi da. Behin programa idatzita, berriro ere probak egingo dira, baina, oraingo honetan, ordenagailuak arazketa automatikoki bideratuko du. Errorerik ezean eta emaitzak egokiak badira, ordenagailu batez exekutagarria den ebazpen-programa sortu da.

Ikus daitekeenez, erabiltzaileek duten problematik abiatuta ebazpen-programara ailegatu behar gara. Gizakiok gure problemak hizkuntza edo lengoaia naturalean adieraz ditzakegu, euskaraz, alegia. Baina ordenagailuek ez dute lengoaia naturala ulertzen. Uler dezaketena lengoaia formalak dira, programazio-lengoaiak, alegia. Beraz, aurreko faseetako urrats bakoitzean problemaren formalizazioa lantzen da (ikus **2. irudia**).



2. irudia. Lengoaia motak eta sistema informatikoarekin lotura

2. gaia

Problema modelizatu: zehaztapena

Problema bat ebazteko, lehenengoz problema analizatu beharra dago. Zer datuk irudikatzen dute gure arazoa? Zein da problemaren soluzioa? Zein dira emaitza lortzeko beharrezkoak? Zergatik dira garrantzitsuak datu hauek eta ez besteak? Nola kodetzen dira datuak? Galdera horiei guztiei erantzuna emateko, **problema abstraitu** egin beharko dugu, hots, problemaren detaileak sinplifikatu, ebatzi nahi diren problemak ahalik eta orokorrenak izateko. Horretarako:

1. **Informazio garrantzitsua identifikatu.** Problema abstraktu horretan, hasierako datuak eta lortu nahi diren emaitzak zein diren identifikatu.
2. **Informazioa kodetu.** «Mundu errealeko» problema kontzeptualki maneiatu eta ebatzi ahal izateko, informazioa datuen bitartez adierazi. Kodetutako **datuak interpretatzen** ditugunean, gure problemari buruz pentsatzen arituko gara. Beraz, problemak baldintzatzen ditu datuak bete beharreko baldintzak zehaztuz.
3. **Problemaren informazioa erlazionatu.** Emaitzak sarrerako datuetatik ondorioztatzen dira; horregatik dira garrantzitsuak. Berriro ere, problemak baldintzatzen ditu datuak haien arteko erlazioak zehaztuz.

Abstrakzioa:	«Errealitate konkretutik abiatuz, horren elementu eta printzipio nagusiak jaso eta irudikatzeko prozesua da». *(Wikipedia)*

Informazio garrantzitsua identifikatu

Problema analizatzeko bera sinplifikatu egin beharko da, soilik ñabardura garrantzitsuak nabarmenduz. Beraz, problemaren emaitzak zein diren identifikatu beharko dira lehenengoz. Ondoren, emaitza hori lortzeko zein datu behar diren identifikatu. Beste datu guztiak soberan egongo dira.

Kontuan hartu, hala ere, gehienetan emaitza lortzeko sarrerako datu multzo desberdinak erabil daitezkeela. Adibidez, hiruki baten azalera lortu nahi bada, oinarriarekin eta altuerarekin kalkula daiteke, baina baita hiru aldeekin ere. Sarrerako datu multzo bat ala beste aukeratu beharko da, eta beste guztiak ahaztu.

Esan bezala, ordenagailuek datuekin egiten dute lan. Programa batean datu asko erabiltzen direnez, beren artean desberdintzeko, bakoitza izen batekin identifikatuko da. Izen esanguratsuak

erabiltzea gomendatzen da interpretazioan laguntzeko. Adibidez, *oinarria* eta *altuera* izenak erabiltzea hobe da, *A* eta *B* erabiltzea baino. Datuak izendatzeko, programazio-lengoaietan jarraitzen diren erregelak hartuko ditugu kontuan. Era honetan, programa sortzerakoan datuen izenak mantendu ahal izango ditugu. Datuak izendatzeko bete beharreko erregelak honako hauek dira:

1. Letra alfabetikoz hasten dira: ‘a’, ‘b’, ‘C’, ‘D’...
2. Ondoren, letra alfabetiko zein karaktere hauek nahas daitezke: ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘_’.

kontagailua2 balizko izena da, baina ez da zuzena *2kontagailua,* adibidez. Kontuan hartu letra txikiak eta larriak ez direla letra berdinak. Hortaz, *kontagailua2* eta *Kontagailua2* izenek datu desberdinak izendatzen dituzte.

Informazioa kodetu

Datuek gure problemaren informazioa kodetzen dute. Alegia, datuek zerbait adierazten dute. Beraz, kodeketarekin batera interpretazioa doa. Adibidez, *azalera* datua zenbakizko balio bat besterik ez da hirukiaren azalera bezala interpretatzen ez badugu. Interpretatzerako orduan, metro karratutan ulertu behar dela jakin beharko genuke.

Problemaren informazioa kodetzeko datuek **oinarrizko balio mota** hauek gorde ditzakete:

— Zenbakizko balioak. **Zenbaki osoak** (zeinu edo zeinu gabe; adibidez –4), eta **zenbaki errealak** (koma hamartarra adierazteko puntu bat ezarriko dugu; adibidez 8.52).
— **Karaktereak**. ASCII taulan (https://eu.wikipedia.org/wiki/ASCII) dauden ikurrak dira. Horietako asko teklatuan aurki ditzakegu. Komatxo sinpleen artean adierazten dira. Adibidez, karaktere alfanumerikoak (‘a’, ‘B’, ‘4’, ...) eta bestelako ikur bereziak (‘&’, ‘@’, ...). Kontuan izan ez direla gauza bera digitu bakarreko zenbaki osoak eta zenbakien karaktereak. Adibidez, 4 balio osoa eta ‘4’ karakterea ez dira berdin irudikatzen, eta kontzeptualki ere ez dira gauza bera.
— **Balio boolearra** edo **logikoa**. Egi eta gezur balioak. Bat dator informatikan erabiltzen den 0 eta 1 kodeketa bitarrarekin. Ingelesezko *true* eta *false* hitzak erabiliko dira kodeketa modura.

Mota honetako balio motekin datu solteak adieraz ditzakegu. Adibidez, «hiruki baten azalera» arazoan emaitza *azalera* datua izango da, zenbaki erreala dena eta metro karratutan ulertu behar dena. Gainera, hiru aldeak izendatu beharko dira: oinarria, ezkerAldea, eta eskuinAldea. Hirurak zenbaki errealak alde bakoitzaren luzera metrotan izanik.

Zenbait problematan, erlazionatuta dauden datu asko izan ditzakegu. Kasu hauetan, datu egituratuak erabiltzea komeni da:

— **Taula homogeneoak**. Dimentsiotan ordenatutako gelaxkez osatuta daude. Gelaxka bakoitza bere posizioaren arabera identifikatzen da, eta balio bakarra gordetzen du. Taula bateko gelaxka guztietan mota berdineko datuak daude. Hau da, zenbaki osoen taula batek gelaxka bakoitzean zenbaki oso bat izango du, inoiz ez beste motako baliorik. Horregatik dira taula homogeneoak.

Esan bezala, zenbaitetan taulek hobeto irudikatzen dituzte problemen datuak. Adibidez, demagun gure asmoa euri-jasen batezbestekoa kalkulatzea dela. Aurreko asteko datuak hauek izan

daitezke: astelehenekoEuria, asteartekoEuria, asteazkenekoEuria, ostegunekoEuria, ostiralekoEuria, larunbatekoEuria eta igandekoEuria. Zazpi eguneko datuak direnez, zazpi datu ditugu. Baina hilabete osoko batezbestekoa kalkulatu nahi badugu? Hogeita hamar datu solte izango ditugu. Urte osoko datuak badira? 365 datu. Ez da eraginkorra hain-beste datu solte izatea. Dimentsio bateko taula batekin egokiago irudika dezakegu arazo honen informazioa. *Euria* datua dimentsio bateko taula izan daiteke. Astebeteko datuak gordetzeko zazpi gelaxka izango ditu, hilabeteko datuak badira hogeita hamar gelaxka, edo urte osoko euri-datuak gordetzeko 365 gelaxka. Baina datu bakarra izango da: *Euria*. Horrelakoa irudika dezakegu taula:

Euria	**Eguna**				
	1	**2**	**...**	**364**	**365**
(l/m ^ 2)	12.5	13.0	...	0.0	1.45

Gelaxken balio errealek euri litroak metro karratuko adierazten dituzte. Gelaxken posizioak aldiz, egunak. Dimentsio bateko taula batek bi kontzeptu erlazionatzen ditu: gelaxken edukiak irudikatzen duena, eta gelaxken posizioak adierazten duena. Adibide honetan, eguna eta egun horretan botatako euria.

Bi dimentsiotako taulek errenkadak eta zutabeak dituzte; beraz, hiru kontzeptu erlazionatzen dituzte: gelaxken edukiak irudikatzen duena, errenkada posizioak adierazten duena, eta zutabe posizioak adierazten duena. Adibidez, instalazio geotermiko egokiena kalkulatzeko, lur azpiko tenperatura-laginak hartzen dira. Demagun etxe baten aurrealdean metro bakoitzeko laginak hartzen direla hamar metrotan. Sakoneran ere metro bakoitzeko lagin bat hartzen da bost metrotan. Beraz, 50 tenperatura-neurketa izango ditugu, eta berrogeita hamar datu soltetan adierazteak ez dirudi egokiena. «Mundu erreala» fisikoki hobekien irudikatzen duen datu-egitura bi dimentsioko taula bat da. Buruan datuen erlazioa era honetan imajina dezakegu:

Tenp		**Metro aurrera**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
metro sakonera	1	15.5	10.0	15.0	10.0	10.0	15.0	10.0	15.5	10.5	18.0
	2	30.0	20.5	25.0	20.0	15.0	20.0	20.5	15.0	10.0	25.0
	3	30.0	20.0	25.5	20.0	15.5	20.0	20.0	15.0	13.5	27.0
	4	32.0	24.0	25.0	22.5	17.0	25.0	21.5	17.5	15.0	27.0
	5	32.5	20.0	28.0	22.5	17.0	25.0	21.0	17.5	15.0	27.5

Tenp datuan berrogeita hamar datuak ditugu bi dimentsioko taula batean, non gelaxken edukia neurtutako graduak baitira, errenkadak sakonera metroak, eta zutabeak aurreranzko metro linealak.

Arestian aipaturiko adibideetan, datuak ez dira mota batekoak, besterik gabe. Benetan problemaren informazioren irudikapena izan daitezen, problemaren testuinguruak inposatzen dituen baldintzak bete beharko dira. Adibidez, hirukiaren aldeak ezin dira negatiboak izan. Aldiz, tenperaturak negatiboak izan daitezke, neguan lur izoztuaren kasuan.

Informazioa erlazionatu

Era berean, problemaren testuinguruak informazioaren arteko erlazioak zehaztu ditzake. Adibidez, hirukietan, edozein bi alderen batura beste aldearen balioa baino handiagoa izango da. Aldi berean, emaitzak beti sarrerako datuekin erlazionatuta egongo dira. Nola kalkulatu bestela emaitzak? Adibidez, euri-jasen batezbestekoa egun bakoitzean botatako litroekin erlazionatuta dago. Izan ere, batezbestekoa litro horien guztien batura zati egun kopurua izango da. Sarrerako datuak emaitzekin erlazionatuz, ebazpena lortzeko prozedura lantzen ari gara hein batean.

Arazo matematiko askotan, formula batek sarrerako datuak emaitzekin erlazionatzen ditu, eta formula hori kalkulatzea izango da problemaren ebazpena. Beste problema mota batzuetan soluzioa ez da hain zuzena izaten. Orduan, sarrerako datuen eta emaitzen arteko erlazioa kasuz kasu definitu beharko da zenbait baldintzaren arabera. Problema ebazteko prozedura konplexuagoa izango da horrelakoetan ere.

Zehaztapena

Esan bezala, problema abstraitu egin behar da informazio garrantzitsua identifikatuz, informazioa datuetan kodetuz, eta datuen arteko erlazioak eta baldintzak definituz. Hori formalki idazteko, **problemaren zehaztapena** egingo da.

Zehaztapena:	«Programa batek hartu behar dituen datuen eta itzuli behar dituen emaitzen deskribapen zehatza. Programa eraikitzeko lehenengo urratsa da». *(Euskalterm)*

Problema bat ondo zehazteko, lortu nahi diren emaitza-datuak eta horiek lortzeko beharrezkoak diren sarrerako datuak deskribatuko dira. Deskribapen zehatz honetan datuak guztiz definitzea da helburua. Datuak nolakoak diren eta beren artean nola erlazionatzen diren adieraziko da. Horretarako, lau galdera hauei erantzungo zaie:

— **Zer datuk definitzen dute problema?** Datuak identifikatu izen esanguratsu eta azalpen baten bitartez. Datu bakoitza nola interpretatu behar den adierazten da. Adibidez, «*oinarria datua hirukiaren oinarria da, metrotan adierazita*».
— **Zein oinarrizko balio mota gordetzen dituzte?** Arestian zerrendatutako oinarrizko balio motaren bat aukeratu beharko da: zenbaki osoak, zenbaki errealak, karaktereak, boolearrak edo horietako datu mota baten taula (taularen egitura adierazi beharko da). Adibidez, «oinarria *zenbaki erreala da*». Hori adieraztearekin badakigu 6,4 balioa gorde dezakeela, baina ez 'a' hizkia. Balio erreal hori metro bezala, eta ez zentimetro bezala, interpretatu beharko litzateke, aurreko galderan adierazi dugunari kasu eginez.
— **Zer murriztapen betetzen dituzte?** Hau da, balio mota batekoa izanda, edozein balio har dezake, ala mugatuta ditu balizko balioak? Baldintzen bitartez adieraziko da zein balio diren onargarriak. Esaterako, balio erreal positiboak onartzen direla adierazteko *oinarria > 0* baldintza ezar dezakegu.
— **Zer erlazio dute datuek beren artean?** Datu mota eta balizko balioak zehaztu ondoren, gerta daiteke datuek beren arteko murriztapenak ezartzea. Sarrerako datuak beren artean egon daitezke erlazionatuta. Adibidez, hiruki baten aldeen arteko erlazioa da: *oinarria < (ezkerAldea + eskuinAldea)*. Irteerako datuak beren artean egon daitezke erla-

zionatuta. Baina, seguru, sarrerako datuak emaitzekin erlazionatuta egongo dira. Izan ere, sarrerako datuak beharrezkoak dira emaitzak kalkulatzeko. Horrela izango ez balitz, ez lirateke sarrerako datuak izan behar; ez genituzke behar gure problema ebazteko.

Era honetan, problemaren ***aurrebaldintza*** (sarrerako datuek bete beharrekoa) eta ***ondorengo baldintza*** (emaitzek bete beharrekoa) zehazten da. Programatzailearen eta **programaren erabiltzaile**en artean nolabaiteko «**kontratua**» ezartzen du zehaztapenak. Kontratu horrek zera dio: programak aurrebaldintzak dioena betetzen duten datuak jasotzen baditu, programak ondorengo baldintza betetzen duten emaitzak kalkulatu eta itzuliko dizkio programaren erabiltzaileari.

Esan bezala, datuak zehazteko datu mota sinpleak erabiliko dira, edo horien taulak. Taulek dimentsioak dituzte: dimentsio bat (bektore), bi (matrize), hiru (kuboa) eta abar. Normalean, bi dimentsiotik gorako taulak ez dira erabiltzen gizakiok buruz irudikatzeko gai ez garelako. Zehaztapenetan, taulak adierazteko datuaren izenaren ondoren dimentsio bakoitzaren tamaina zehaztuko da. Adibidez, «*Tenperaturak (3,4): zenbaki errealen taula*» idatziko da, *Tenperaturak* izeneko datuak 3 errenkada eta 4 zutabe dituela eta gelaxka bakoitzean zenbaki erreal bat gordeko duela adierazteko. Gogoratu tauletako gelaxka guztiak mota berdinekoak direla, taulak homogeneoak direla, alegia.

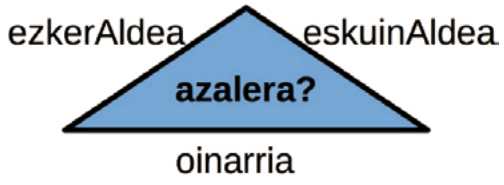
Ahalik eta zehatzenak izan behar dute zehaztapenek. Ahal bada lengoaia logiko-matematikoa erabiliko da; unibertsala, ulergarria eta unibokoa den heinean. Hala ere, ezinbestean, euskara xumean adieraziko da zehaztapena. Ezagutzen dugun lengoaia matematikoa erabiliko da: eragiketak (*, –, batukaria, ...)[2] eta erlazioak (<, <=, ==, ~=....)[3]. Logika formalaren zenbait eragile ere erabiliko ditugu baldintzak adierazteko (edo, eta, ez).

Beraz, zehaztapena, problemaren modelo bat da. Hau da, argi eta garbi adierazten da problema zer den eta datuak nola interpretatu behar diren. Hemen, adibideak aurkezten dira, atal honetako ideiak eta kontzeptuak lantzeko.

1. adibidea. *«Hiruki baten azalera kalkulatu nahi da bere hiru aldeak emanda»*

Enuntziatu horrek hitzez adierazten du ebatzi nahi den problema. Guk, programatzaile bezala, zehaztapena egingo dugu problema nola ulertzen dugun ezartzeko. Ondoren, gure programak zehaztapenak dioena ebatzi beharko du.

Buruan, horrela irudika dezakegu problema:



[2] Informatikan biderketa izartxoaren (*) bitartez adierazten da, eta ez *x* ikurrarekin; izan ere, *x* datu baten izena izan daiteke.
[3] Programazio-lengoaietan = ez da berdintasun-konparaketa izaten, baizik eta esleipena. Berdintasun-konparaketa == adieraziko dugu, eta desberdintasuna ~= .

Enuntziatua ez al da jada problemaren definizioa? Hein batean bai, baina enuntziatuan informazioa falta eta kontraesanak egon daitezke. Zehaztapena, aldiz, guztiz argia izango da. Horretarako, enuntziatuak dioena kontuan hartuz, **datuak zehazteko erabakiak hartu beharko ditugu**. Lehen erabakia triangeluaren aldeak zein motatakoak diren eta zein unitatetan emango diren zehaztea izango da. Hartzen ditugun erabakiek eragina izango dute problemaren ebazpenean (eta, ondorioz, programan). Demagun erabakitzen dugula aldeak metrotan ematen direla, nazioarteko sistema metriko hamartarrak horrela ezartzen duelako. Aldi berean, balio osoak izatea erabaki dezakegu. Erabaki horiek duten eragina aurreikusi behar dugu. Triangelu posible guztiak adieraz al daitezke metrotan, zenbaki osoak izanik? Ez! 1,5 metroko aldea duen triangelu bat ezin dugu adierazi. Beraz, gure erabakia ez da egokia triangelu posible guztien azalera kalkulatu nahi badugu.

Programa ahalik eta orokorrena izaten saiatu beharko gara. Diseinu eta programazio-lan bat egingo badugu, ahalik eta erabilgarriena izan dadila programa. Gure adibide honetan, programak hiruki posible gehienen azalera kalkula dezala, alegia. Hori lortzeko, aldeak ez direla metrotan interpretatu behar erabaki dezakegu, baizik eta zentimetrotan. Era horretan, 1,5 metroko aldea 150 zentimetrotan adieraz dezakegu. Orain, triangelu posible guztiak adieraz al daitezke zentimetrotan zenbaki osoak izanik? Ez! 1,5 zentimetroko aldea duen triangelua ezin dugu adierazi. Problema nagusia da zenbaki osoetan ezin direla neurri guztiak adierazi. Erabaki zentzudunena zenbaki errealak erabiltzea izango litzateke. Orain bai, edozein neurritako triangeluak adieraz daitezke.

Zehaztapenean, sarrerako zein irteerako datuak definitu behar dira. Irteerako datua argia da, azalera, hori baita gure probleman lortu nahi dugun emaitza. Gogoratu, **emaitzak beti egongo dira erlazionatuta sarrerako datuekin, sarrerako datu horien beharra nabarmenduz.** Jakinda hiruki baten azalera kalkulatzeko formula bat hau dela: *azalera = (oinarria * altuera) / 2*; esan dezakegu sarrerako datuak *oinarria* eta *altuera* direla. Alabaina, ezin gara joan enuntziatuak dioenaren kontra. Enuntziatuak espresuki aipatzen ditu: «hiru aldeak emanda». Beraz, *altuera* datua ez da sarrerako datua, baizik eta hiru aldeak: *oinarria*, *ezkerAldea* eta *eskuinAldea*.

Sarrerako datuak hiru aldeak baldin badira, horietatik abiatuz azalera nola lor dezakegun pentsatu beharko da. Formula bat bada hori egiten duena, *Heron*en formula: *azalera = s * (s - oinarria) * (s - ezkerAldea) * (s - eskuinAldea)*. Non *s* perimetroerdia baita: *s = (oinarria + ezkerAldea + eskuinAldea) / 2*. Hortaz, identifikatu ditugu sarrerako eta irteerako datuak, eta nola erlazionatzen diren. Zehaztapena argi gera dadin, hemen agertzen den moduko taula bat betetzen da:

	Aurrebaldintza (sarrerako datuak)	Ondorengo baldintza (emaitzak)
Zer	*oinarria: hirukiaren oinarria metrotan.* *ezkerAldea: ezkerreko aldea metrotan.* *eskuinAldea: eskuineko aldea metrotan.*	*azalera: hirukiaren oinarria metrotan.*
Mota	*oinarria: zenbaki erreala.* *ezkerAldea: zenbaki erreala.* *eskuinAldea: zenbaki erreala.*	*azalera: zenbaki erreala.*
Mugak	*oinarria > 0 eta ezkerAldea > 0* *eta eskuinAldea > 0*	*azalera > 0*
Erlazioak	*oinarria < (ezkerAldea + eskuinAldea)* *eta ezkerAldea < (oinarria + eskuinAldea)* *eta eskuinAldea < (oinarria + ezkerAldea)*	$azalera = \sqrt{s * (s - oin) * (s - ezA) * (s - esA)}$ *non:* *s = (oin + ezkerAldea + eskuinAldea) / 2* *oin= oinarria* *ezA= ezkerAldea* *esA= eskuinAldea*

2. adibidea. *«Zenbaki baten faktoriala»*

Zenbaki bat emanda, bere faktoriala kalkulatu nahi da. Alegia, erabiltzaileak 5 balioa ematen badu 120 emaitza jaso nahi du, zeren eta *5! = 5 * 4 * 3 * 2 * 1 = 120* baita.

Zehaztapen bat egiten dugunean, ahal diren kasu gehienak kontuan hartzea komeni da. **Zenbat eta denbora gehiago inbertitu zehaztapenean, ustekabe gutxiago izango dugu programan**. 1. IRUDIAk erakusten duen bezala, ustekabe bat konpontzeak zehaztapena berregitea suposa dezake, gure lana lehenengo pausotik hastera behartuz. Beraz, **sarrerako datu posible guztiek sor ditzaketen arazoak aurreikustea komeni da denbora aurrezteko**. Baina, infinitu kasu posible dira, eta ezin dira guztiak banan-banan kontuan hartu. Multzoka ditzakegu, ordea. Multzo kopurua finitua izango da: positibo-negatibo, bikoiti-bakoiti, 3ren anizkoitzak, 100 baino txikiagoak direnak... Problema bakoitzean kasu multzo interesgarriak identifikatu beharko dira.

Faktorialaren ariketa honetan, 5 balioaren faktoriala existitzen dela ikusi dugu. Beraz, ondoriozta dezakegu zenbaki oso positiboen faktoriala arazorik gabe kalkula daitekeela. Hau da, 4 edo 6 edo 129 balioen faktoriala existitzen dela. Baina, zenbaki oso negatiboen faktoriala kalkula al daiteke? Negatiboen multzoko ordezkari bat hartuta baieztatu beharko da ea bere faktoriala existitzen den. Har dezagun –5 balioa. Bere faktoriala –120 dela dirudi. Ikus dezagun, *–5! = –5 * –4 * –3 * –2 * –1 = –120*. Horrela dirudi, baina ez da zuzena.

Matematikan (eta beste arlo batzuetan), aldez aurretik finkatutako eragiketen definizioa aztertzea komeni da. Faktorialaren definizioak zehazten duena jarraitu beharko dugu, bestela, agian, faktoriala ez den beste eragiketa bat ebatziko dugu eta. Ez dugu erabaki beharrik faktoriala nola ulertzen dugun, baizik eta definizioak zehazten duena onartuko dugu. Faktoriala Christian Kramp

(1760ko uztailaren 8an jaioa, 1826ko maiatzaren 13an zendua) matematikari frantsesak asmatu zuen.

Faktoriala:	«Faktorial bat edozein n oso naturala hartuta 1 eta n artean dauden zenbaki guztien biderkaduraren emaitzari deitzen zaio». *(Wikipedia)*

Uste genuen bezala, faktoriala era honetan definitzen da: n! = n * (n – 1) * * 3 * 2 * 1. Horregatik, 5 faktoriala 120 da. Gainera, definizioak n zenbaki naturala (zenbaki arruntak bezala ere ezagunak) izan behar dela zehazten du. Hortaz, zenbaki negatiboen faktoriala ez da existitzen, ezta zenbaki errealena ere. Beraz, ezin da 5.5 zenbakiaren faktoriala kalkulatu. 5.5 * 4.5 * 3.5 * 2.5 * 1.5 * 0.5 = 162.42 kalkula daiteke, baina ez da faktoriala, beste eragiketa bat da. «*Faktoerrela*» eragiketa asmatu dugu!

Orduan, definizioa aztertuz argitu dugu faktoriala eragiketa zenbaki oso eta positiboena dela soilik. Beste kasu berezi bat ere aztertu beharra dago: zero balioa. Zero balioak askotan portaera berezia izaten du eragiketa matematikoetan; horregatik, kontuan hartu beharreko kasu berezi bat izaten da. 0 balioaren faktoriala kalkula daiteke, baina, zein da? 1 ala 0? Ez dugu gehiegi pentsatu behar; definizioz, zero balioaren faktoriala 1 balioa da.

Kasu orokorrak eta bereziak aztertu ostean, zehaztapena idatz dezakegu.

	Aurrebaldintza	**Ondorengo baldintza**
Zer	*n: faktoriala kalkulatzeko balioa.*	*Emaitza: sarrerako n balioaren faktoriala.*
Mota	*n: zenbaki osoa.*	*Emaitza: zenbaki osoa.*
Mugak	*n >= 0*	*Emaitza > 0*
Erlazioak		*Baldin n == 0 bada, orduan* *Emaitza == 1* *Baldin n > 0 bada, orduan* *Emaitza == n * (n-1)!*

Zehaztapen horrek kontutan hartzen ditu lehenago pentsatu ditugun kasu guztiak. n ezin da erreala izan, ezta negatiboa ere. Emaitza sarrerako balioaren faktoriala izango da. n! eragiketa matematiko ezaguna den arren, argiago azalduko dugu. n datuaren balioa zero denean, bere faktoriala 1 da. n datuaren balioa zero baino handiagoa denean, emaitza n bider (n – 1) balioaren faktoriala izango da. Hizkuntzalaritzan definitzen ari garen kontzeptua ezin da definizioan ageri (adibidez, «euria da lainoetatik erortzen den euria»), baina matematikan bai: *funtzio errekurtsibo* deritzo. Baina zuzena izan dadin errekurtsioa, emaitza ematen duen oinarrizko kasu batera ailegatu behar da. Kasu honetan, egoki definituta dago. Ikus dezagun. Demagun n datuak 5 balioa

duela. Definizio errekurtsiboaren arabera, bere faktoriala da 5 * (5-1)!. Emaitza lortzeko 4! kalkulatu beharko da: 4 * (4-1)! izanik. Hori da errekurtsioa, behin eta berriro faktoriala kalkulatzeko faktoriala bera kalkulatu behar izatea. Errepikapen hori noizbait bukatu beharko da emaitza lortzeko. Errekurtsioan kalkulatu beharreko faktoriala unitate batean txikiagotzen doa. 4, 3, 2, 1 eta, azkenean, 0 izango da. Hau da oinarrizko kasua: 0! da 1 balioa. Beraz, 5! da 5 * 4 * 3 * 2 * 1 * 1. Emaitza, 120.

3. adibidea. *«10 metroko U motako instalazio geotermiko bertikala»*

Formula matematiko batez ebazten diren problemak baino askoz konplikatuagoak direnak izan ditzakegu. Adibidez, instalazio geotermiko bertikal baten instalazio eraginkorrena kalkulatu nahi badugu. Hau da, lur azpiko berotasuna hobekien harrapatzen duen hodien instalazioa. Hodiek «U» forma izango dute derrigorrez, eta gehienez 10 metroko luzera, kostu ekonomikoa dela eta.

Lur azpiko tenperaturen laginak ditugu metroz metro, 10 metro gure etxetik aurrera, eta beste 5 metro lur behera; beraz, 50 datu (5 lur beherako metroak bider 10 etxe aurreko metroak). Datu horiek taula batean izan ditzakegu; errenkadak lur azpiko metroak adieraziz, eta zutabeak etxe aurreko metroak. Honako taula honetan, balioak adibide modura ematen dira:

Tenp		**Metro aurrera**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
metro sakonera	1	15.5	10.0	15.0	10.0	10.0	15.0	10.0	15.5	10.5	18.0
	2	30.0	20.5	25.0	20.0	15.0	20.0	20.5	15.0	10.0	25.0
	3	30.0	20.0	25.5	20.0	15.5	20.0	20.0	15.0	13.5	27.0
	4	32.0	24.0	25.0	22.5	17.0	25.0	21.5	17.5	15.0	27.0
	5	32.5	20.0	28.0	22.5	17.0	25.0	21.0	17.5	15.0	27.5

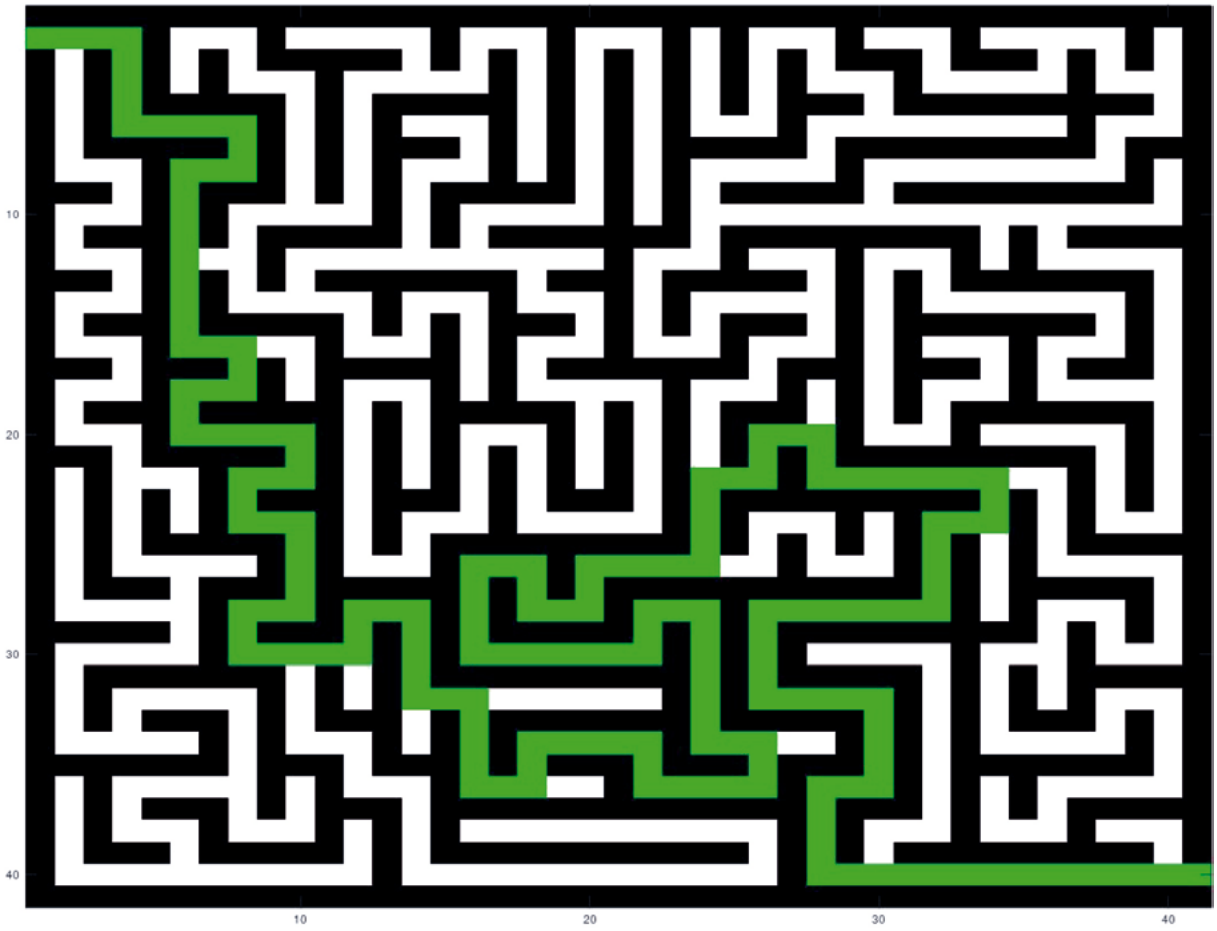
Problemaren ebazpena instalazio eraginkorrena definitzen duten datuak izango dira. Hau da, hodia zein metrotan lurperatzen den (sarrera), zenbat metro behera egiten duen (sakonera) eta nondik irteten den (irteera) jakin behar dugu, gehienez 10 metroko luzerako murriztapena errespetatuz. Baina ez du edozein instalaziok balio; instalazio eraginkorrena izan behar du, hots, berotasun gehien harrapatzen duen instalazioa. Hodi barneko likidoak hodia pasatzen den tokiko berotasun guztia hartzen duela suposatuko dugu. Termodinamikak ez du modelo hori onartzen, besteak beste, berotasun-galerarik kontuan hartzen ez duelako; baina sinplifikazio horrek instalazio eraginkorrena kalkulatzeko balio digu.

Problema horren zehaztapena hau izan daiteke:

	Aurrebaldintza	Ondorengo baldintza
Zer	*tenp: laginen tenperatura-taula*	*sarrera: etxe aurreko zein metrotan sartu hodia.* *irteera: etxe aurreko zein metrotan atera hodia.* *sakonera: zenbat metro lurperatu.*
Mota	*tenp(5,10): zenbaki errealen taula*	*sarrera: zenbaki osoa.* *irteera: zenbaki osoa.* *sakonera: zenbaki osoa.*
Mugak		*sarrera >= 1 eta sarrera <= 9 eta irteera >= 2 eta irteera <= 10 eta sakonera >= 1 eta sakonera <= 5*
Erlazioak		*((irteera - sarrera) + sakonera * 2) <= 10 eta sarrera < irteera eta* $$xurgatua = beherantz + aurrerantz + gorantz$$ *xurgatua guztien artean maximoa. Non:* $$beherantz = \sum_{err=1}^{sakonera} tenp\,(err,sarrera)$$ $$aurrerantz = \sum_{zut=sarrera+1}^{irteera-1} tenp\,(sakonera,zut)$$ $$gorantz = \sum_{err=1}^{sakonera} tenp\,(err,irteera)$$

Zehaztapenak dioenaren arabera, problemaren datu-sarrera 5 × 10 balio erreal dituen taula bat da. Hamar metroko luzeran eta bost metroko sakoneran hartutako laginen balioak dira. Balio erreal guztiak onartzen dira, mugarik gabe. Hau da, tenperatura negatibo bat (lurra izoztuta dagoenean) balizko balioa da. Ondorengo baldintzak zehazten duenez, emaitza hiru datuz osatua dago: hodiaren sarrerako kokapena, hodiaren irteerako kokapena, eta zein sakonera arte lurperatu U motako hodia. Bakoitzak bere murriztapenak ditu. Gainera, ez da onartzen edozein <*sarrera*, *irteera*, *sakonera*> konbinazio; sarrerako zuloaren posizioak irteerakoaren aurretik egon behar du, gehienez 10 metroko hodia erabil daiteke eta xurgatutako tenperatura maximizatzen duen konbinazioa izan behar da. Hori adierazteko, batukariak erabiltzen dira taulako balio desberdinak batzeko. Adibidez, *tenp (err, sarrera)* izango litzateke *tenp* taularen balio bat, *err* errenkada eta *sakonera* zutabean dagoena.

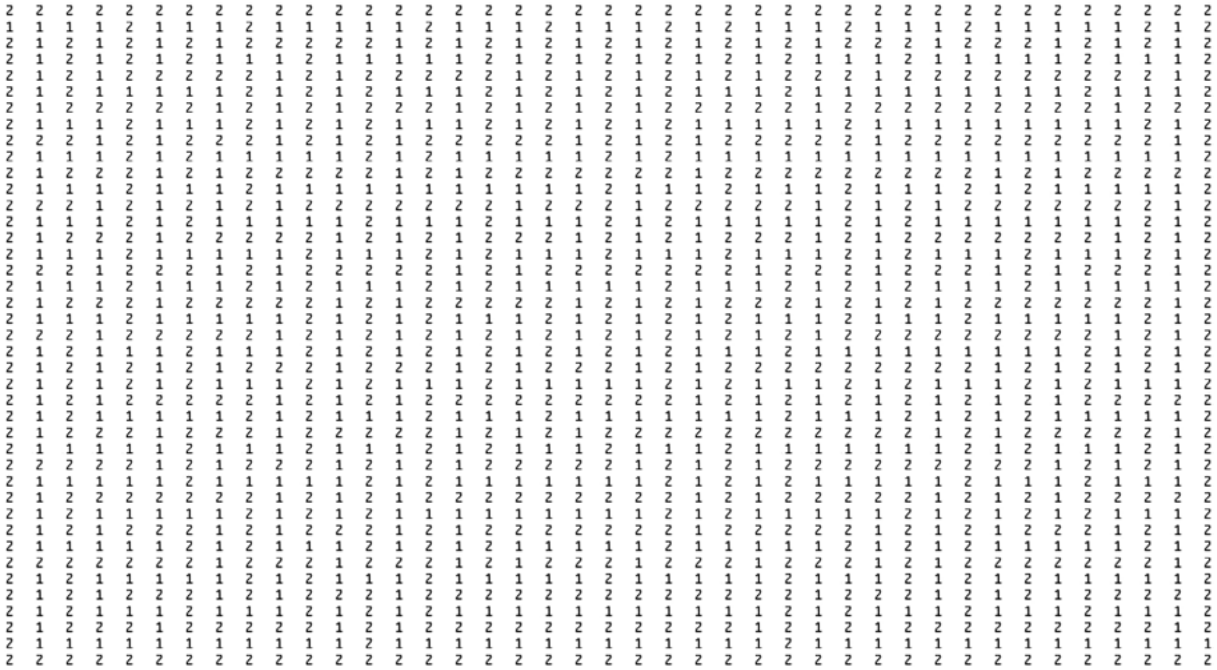
4. adibidea. *«Labirinto baten irteera»*



Kasu honetan, problemaren sarrera labirintoaren mapa bat izango da, eta emaitza mapa bera irteerako bidea markatuta duela. Labirinto mota asko dagoenez, murriztapen hauek ezarriko ditugu:

— Bide guztiek unitate bateko zabalera dute.
— Ez dago begiztarik (biribilgunerik).
— Sarrera bakarra dago, goi-ezker izkinean.
— Irteera bakarra dago, behe-eskuin izkinean.
— Bide bakarra dago, sarreratik irteerara doana.

Guk gizakiok, problemez errazago pentsatzeko, mundu fisikoa modelizatu egiten dugu. Labirinto fisikoa koloreztatutako mapa bihurtzen dugu: zuria bidea, beltza pareta. Ondoren, bide egokia bilatzen hasten gara mapa horretan. Era berean, **ordenagailuek mundu fisikoaren modeloa kodetuta jaso behar dute datu modura, ondoren datu horiek prozesatu eta soluzioa kalkulatzeko.** Izan ere, ordenagailuek oinarrizko balioekin egiten dute lan (zenbaki, karaktere eta balio logikoak). Labirintoaren mapa eskaneatuta eman diezaiokegu ordenagailuari, baina orduan soluzioa zaildu egiten da, irudi-prozesaketarako programa bat beharko genukeelako. Hobe da ordenagailuari labirintoa zenbaki-taula modura kodetuta ematea. Hau izan daiteke kodeketa: esploratu gabeko bidea 1 balioa, pareta 2 balioa. Honako hau da taula horren adibide bat:



Aldi berean, prozesaketa egiteko programak ere kodeketa hori erabil dezake: bide egokia 3 balioa.

Labirintoaren zehaztapena honako hau izan daiteke:

	Aurrebaldintza	**Ondorengo baldintza**
Zer	*labirintoa: kodetutako labirintoaren mapa.*	*labirintoa2: kodetutako labirintoaren mapa.*
Mota	*labirintoa (n, m): zenbaki osoen taula.*	*labirintoa2 (n, m): zenbaki osoen taula.*
Mugak	*n > 1 eta m > 1eta labirintoa (j,k) ∈ {1,2} eta 1<=j<=n eta 1<=k<=m eta labirintoa (2,1) = 1 eta labirintoa (n-1,m) = 1 eta* *bide guztiek unitate bateko zabalera dute,* *ez dago begiztarik (biribilgunerik),* *bide bakarra dago, sarreratik irteerara doana.*	*n > 1 eta m > 1eta labirintoa2 (j,k) ∈ {1,2} eta 1<=j<=n eta 1<=k<=m eta labirintoa2 (2,1) = 3 eta labirintoa2 (n-1,m) = 3 eta tarteko bidea 3 balioaz markatuta dago.*
Erlazioak		*labirintoa2 labirintoa bezalakoa da, baina sarrera, hau da, labirintoa2 (2,1) gelaxka, eta irteera, hau da, labirintoa2 (n-1,m) gelaxka, 4 balioez markatutako bide batekin lotuta daude.*

3. gaia

Problema ebatzi: algoritmoa

Zehaztapenak problema definitzen du, problemaren hasierako datuak eta emaitzak zehaztuz. Hau da, problema modelizatzen du. Algoritmoak hasierako datu horietatik emaitzak nola lor daitezkeen zehazten du. Hau da, problema ebazteko prozedura zehazten du. Zehaztapenaren eta algoritmoaren arteko erlazioa estua da. Algoritmoa zehaztapenaren ikuspegitik ulertu behar da. Izan ere, algoritmoak zehaztapenean ageri diren datuekin lan egingo du. Halaber, algoritmoak kontuan hartu beharko ditu zehaztapenean datuei ezarritako baldintzak. Nolabait, **zehaztapena kontratu moduko bat da**: algoritmoak zehaztapenaren aurrebaldintzan ezarritako baldintzak betetzen dituzten datuak jasotzen baditu, algoritmoak ondorengo baldintzan ezarritako baldintzak betetzen dituzten emaitzak bueltatuko ditu. Ondo ulertu behar da kontratua; aurrebaldintza betetzen ez duten datuak jasoz gero, algoritmoak ez du zertan ondorengo baldintza betetzen duten emaitzak bueltatu beharrik.

Beraz, algoritmoak zehaztapenean ageri diren datuekin lan egingo du. Prozesaketa egin ahala, datuen balioak alda daitezke; horregatik, programetan datuei **aldagai** deritze. Daturen batek finkoa izan behar baldin badu, **konstante** deritzogu. Adibide bat *PI* konstante matematikoa izan daiteke.

Algoritmoa programaren diseinua da, ondoren programa bihurtzen dena. Ordenagailuek, gizakiok ez bezala, ez dituzte aginduak interpretatzen, zehatz-mehatz agindutakoa betetzen dute. Horretarako, aginduek argiak izan behar dute; hau da, algoritmoek ezaugarri hauek izan behar dituzte:

— Zehatzak izan behar dute.
— Pausoen arteko ordena azaldu behar dute.
— Beti amaitu behar dute.
— Sarrerako datu berdinekin, bi aldiz algoritmoak dioenari jarraituz, emaitza berdina lortu behar da.
— Programazio-lengoaiarekiko independenteak izan behar dute.

Algoritmoak zehatzak izan daitezen, **programazio inperatiboa** erabiliko dugu; hau da, **agindu multzo ezagun eta mugatu bat** erabiliko dugu. **Sarrerako datuak ordenagailura sartu, ondoren datuak prozesatu eta, azkenik, emaitzak ordenagailutik ateratzeko balio dute aginduek**. Agindu horiek ordenagailuarentzat ulergarriak dira, eta algoritmoak adierazten duen ordena jarraituan egikarituko ditu ordenagailuak, bata bestearen ondoren. Era horretan, sarrerako balio berdinekin emaitza berdina lortuko da beti.

Baina, zenbait problema ezin dira ebatzi agindu kopuru zehatz bat egikarituta. Adibidez, erabiltzaileak ematen duen 0 balioaz bukatzen den zenbaki zerrenda baten batura kalkulatu nahi badugu, zenbat aldiz eskatu behar du algoritmoak zerrendaren hurrengo zenbakia? Hamar zenbaki izango balira, nahiko izango litzateke hamar balio eskatu eta batzearekin. Tamalez, aldez aurretik ezin daiteke jakin erabiltzaileak zenbat zenbaki emango dituen.

Horrelako problemak ebazteko **programazio egituratua** erabiltzen da. **Kontrol-egitura berezi batzuk** erabiliz, baldintzen arabera, aginduen exekuzio jarraitua alda dezakete. Alde batetik, **egitura errepikakorrak izango ditugu,** agindu multzo bat behin eta berriro egikaritzen dutenak baldintza bat betetzen den bitartean. Bestetik, **baldintzapeko adarkatze-egiturak,** agindu multzo bat egikarituko den ala ez erabakitzen dutenak baldintza baten arabera. Adibidez, lehen adierazitako problema datu berri bat eskatuz eta batuz ebatz daiteke, sarrerako datua zero den arte. Hori egitura errepikakor batek egin dezake.

Kontura gaitezen kontrol-egiturak erabiliz algoritmoak asko labur daitezkeela. Ehun zenbaki batu behar baldin baditugu, ehun aldiz balio berri bat eskatu eta batu beharrean, kontrol-egitura errepikakor baten bitartez errepika dezakegu. Kontagailu baten bitartez, zenbat zenbaki batu diren kontrola daiteke (ehun kontatu arte). Algoritmoa motzagoa eta ulergarriagoa izango da.

Hala eta guztiz ere, algoritmoak asko luza daitezke. Horregatik, **programazio modularra** ere erabiltzen da. Programazio mota horretan algoritmoak modulu bihurtzen dira, beste algoritmoetan berrerabiliak izan daitezen. Aldez aurretik egindako algoritmo berrerabilgarri hauei *modulu*, *azpialgoritmo* edo *azpiprograma* deritze. Abantailak hauek dira:

— Problema handi bat problema txikiagotan zatituz errazago ebatz daiteke.
— Algoritmoak laburragoak eta ulergarriagoak dira.
— Behin egin eta askotan erabiliz, denbora aurreztu eta akats gutxiago sortzen dira.

Programazio inperatiboa

Programazio inperatiboan aginduek eragina dute aldagaietan, normalean, balioa aldatuz. Matematikan X = 5 jartzen denean, egia bat esaten ari gara: «X aldagaiak 5 balioa du». Aldiz, programazioan X = 5 jartzen dugunean, ordenagailuari agintzen diogu X aldagaian 5 balioa gordetzeko. Esleipen-agindua da, X aldagaiaren balioa aldatu egiten da. Ondoren, algoritmoetan (eta programazioa-lengoaia gehienetan) erabil ditzakegun aginduak zerrendatuko ditugu. Kontuan izan algoritmo eta programazio-lengoaietan aginduak idazteko modua oso zehatza dela, eta bera bete behar dela. Idazkera edo sintaxi zuzena jarraitu behar dela, alegia.

Esleipena

Aldagaiek momentu bakoitzean balio bakarra gordetzen dute. Baina balio hori alda daiteke esleipena eragiketaren bitartez. '=' ikurrak esleipena adierazten du, eta bere eragina da bere ezkerraldean dagoen aldagaiak eskuinaldeko balioa hartzen duela. Sintaxia horrelakoa da: ***<ALDAGAIAREN_IZENA> = <BALIOA>***[4].

[4] Eskuliburuan zehar, < eta > ikurren artean agertzen direnak ordezkatu beharreko kontzeptuak direla ulertu behar da. Adibidez, <*BALIOA*> jartzen duenean, balio zehatz bat izango dugula adierazi nahi da.

Aldagaia taula baldin bada eta gelaxka bakarra aldatu nahi bada, gelaxka identifikatu beharko da bere taula barneko posizioa adieraziz. Adibidez, bektore bat bada (dimentsio bakarreko taula), era honetan egingo dugu: ***<ALDAGAIAREN_IZENA> (<POSIZIOA>) = <BALIOA>***. Parentesiak adierazten du bektoreko zein posiziotan gorde behar den balioa. Aldiz, bi dimentsioko taula batean, ***<ALDAGAIAREN_IZENA> (<ERRENKADA_ZENBAKIA>, <ZUTABE_ZENBAKIA>) = <BALIOA>*** jarri beharko da. Taulen posizioak 1 zenbakitik aurrera hasten dira zenbatzen. Adibide modura, jo dezagun egun bateko orduz orduko tenperaturak gordeta ditugula *tenperatura* bektorean. Eguerdiko hamabietan 20 gradu izan direla gordetzeko, era honetan aginduko genuke: *tenperatura(12) = 20*.

Esleitzen den balioa mota hauetako bat izan daiteke: balio konstanteak, balioen taula, aldagai baten balioa ala eragiketa baten emaitza. Modulu edo azpiprogramen emaitza ere eslei daiteke, baina horiek beste atal batean landuko dira.

Balio konstanteak oinarrizko datu moten balio ezagunen irudikapenak dira: zenbaki errealak (adibidez, *5.2*), zenbaki osoak (5), balio logikoak (egia-bai-true eta gezurra-ez-false) eta karaktereak (‘a’, *‘b’ eta abar*).

Balioen taulak ere eslei daitezke. Taula baten balioak definitzeko, balioak ‘[‘ eta ‘]’ ikurren artean jarriko dira. Adibidez, ***nireBektorea= [1 2 3 4]*** aginduak *nireBektorea* aldagaian lau gaien bektorea gordetzen du. Ulertzen da lehen gaia lehen posizioan dagoela (1 balioa adibide honetan), bigarren gaia bigarren posizioan, eta horrela hurrenez hurren. Taulak bi dimentsio baditu, ‘;’ ikurrez desberdinduko dira errenkadak. Adibidez, ***nireTaula = [10 20 30; 40 50 60]*** esleipenak bi dimentsioko taula gordetzen du *nireTaula* aldagaian. Lehen errenkadan 10, 20 eta 30 balioak ditu, bigarrenean 40, 50 eta 60 balioak.

Aldagai baten balioa beste aldagai bati esleitzeko, = ikurraren eskuinaldean aldagaia adierazi beharko dugu. Adibidez, eguerdiko ordu batean hamabietan izan den tenperatura berdina bada, era honetan gorde dezakegu balio hori: *tenperatura(13) = tenperatura(12)*. Adibide honetan ikus daiteke balioa jasotzen duen aldagaia taula baten gelaxka izan daitekeela baita ere.

Eragiketa baten emaitza aldagai bati esleitzeko, lehendabizi eragiketak egikaritu beharko dira. Jarraian, azaltzen dira algoritmoetan erabil daitezkeen eragileak.

Prozesaketa

Sarrerako datuetatik emaitzak nola lortzen diren zehazten dute algoritmoek. Horretarako, sarrerako balioekin kalkuluak egin beharko dira. Eragile matematiko eta logikoak dira datuak eraldatzen dituztenak. Ohiko **eragile matematiko**ak hauek dira:

— batuketa: +
— kenketa: –
— biderketa: *
— zatiketa: /
— berreketa: ^

Eragiketa matematikoek bi zenbaki hartu eta beste zenbaki bat bueltatzen dute emaitza modura. Eragile erlazionalek bi balio (zenbaki ala karaktere) hartu eta balio boolearra bueltatzen dute (bai ala ez balioak). **Eragile erlazional**ak hauek dira:

— handiago: >
— txikiago: <
— berdin: ==
— desberdin: ~=
— handiago edo berdin: >=
— txikiago edo berdin: <=

Lokarri logikoek bi balio logiko hartu eta beste balio logiko bat bueltatzen dute. Honako hauek ditugu eskuragarri:

— edo: ||
— eta: &&
— ukapena: ~

Aurreko eragile guztiak bitarrak dira, eta eragingaien artean kokatzen dira. Ukapena eta minus zeinua izan ezik. Horiek ukatu edo zeinu-aldaketa eragin behar zaion eragingaiaren aurrean kokatzen dira. Honako adibide hauek erakusten dute nola erabiltzen diren aurreko eragile batzuk:

— *Batura = 5+3*; emaitza 8 da, eta *Batura* aldagaian gordetzen da esleipenaren bitartez.
— *BatezBestea = Batura / 2*; aurreko aldagaiaren balioa hartuz, emaitza 4 da.
— *HandiagoaAlDa = BatezBestea > 5*; aurreko aldagaiaren balioa hartuz, emaitza *false* da.
— *TxikiagoaBerdinaAlDa = ~ HandiagoaAlDa*; aurreko aldagaiaren balioa hartuz, emaitza *true* da.

Sarrera/irteera aginduak

Sarrerako aginduek balioak jasotzen dituzte, eta esleipenaren bitartez aldagai batean gorde. Algoritmoetan, ***IRAKURRI*** aginduak jasotzen ditu balioak. Nondik jasotzen diren adierazi beharko da:

— Fitxategi bateko balioak hartzen badira, fitxategia identifikatzen duen izena adierazi beharko da. Adibidez, *tenperaturak = IRAKURRI ('tenperaturak.txt')*.
— Ohikoena erabiltzaileak teklatuan sakatutako balioa hartzea da. Orduan, IRAKURRI aginduan ez da ezer adierazten. Alegia, balioa nondik jasotzen den adierazten ez bada, teklatua da datu-sarbidea. Adibidez, *adina = IRAKURRI ()*.

Datu-irteera ***IDATZI*** aginduarekin egiten da. Agindu horrek ez du aldagaien baliorik aldatzen, baizik eta balioak erakutsi edo gorde.

— Fitxategi batean idazteko fitxategia bere izenez identifikatuko dugu. Zein aldagai edo balio gorde nahi den ere adieraziko da. Adibidez, *IDATZI ('tenperaturak.txt', tenperaturak)*.
— Taulak grafiko modura erakuts daitezke. Kasu honetan, leiho grafikoaren identifikatzailea adierazi beharko dugu. Adibidez, *IDATZI ('tenperaturenGrafikoa', tenperaturak)*.
— Ohikoena erabiltzaileari informazioa pantailaz erakustea izaten da. Kasu honetan, pantailaratu beharreko balioa edo aldagaia adierazten da soilik. Adibidez, *IDATZI ('Kaixo. Zer adin duzu?')*.

Programazio egituratua

Algoritmoak adierazteko era asko dago. Eskuliburu honetan *fluxu-diagramak* erabiliko ditugu. Exekuzio-fluxua grafikoki adierazten da diagrama batekin. Elementu grafiko ezagun batzuk geziekin lotzen dira, egikaritze-ordena adieraziz. Fluxu-diagrametan erabil ditzakegun elementu grafikoak hauek dira:

— **Terminalek** exekuzio hasiera eta bukaera adierazten dute:

— **Sarrerako eta irteerako** aginduak adierazteko laukia:

— **Esleipen-aginduak** adierazteko laukizuzena:

— **Baldintzak** erronbotan adierazten dira:

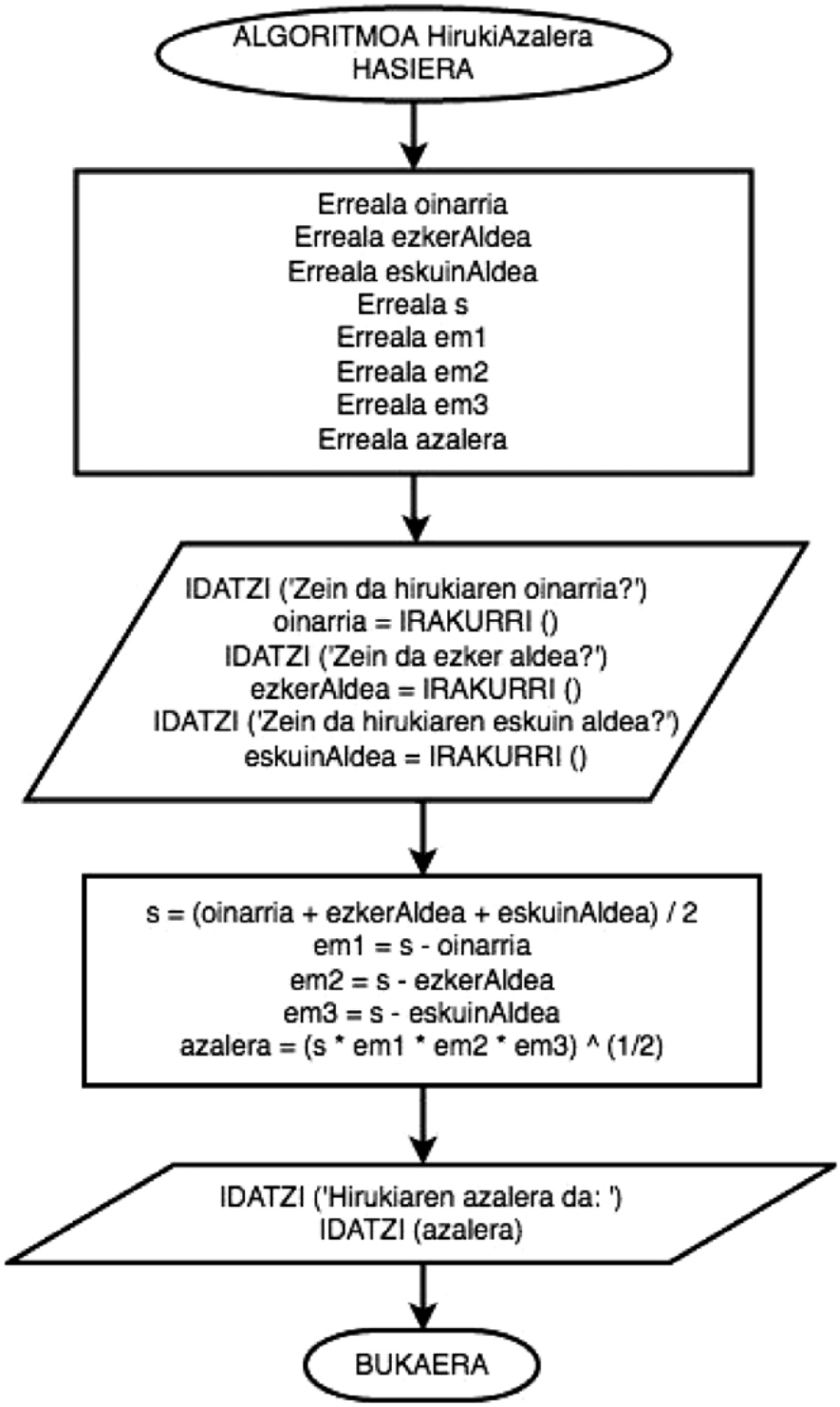
Algoritmoaren exekuzio-fluxua hasierako terminaletik bukaerako terminalera arte izango da, geziek zehazten duten bidea jarraituz. Algoritmoen hasieran erabiliko diren aldagaiak erazagutzen dira. Laukizuzen baten barruan aldagaien izena eta mota ematen dira ezagutzera. Zehaztapenean adierazitako sarrerako eta irteerako datuak izango ditugu. Horiez gain, tarteko emaitzak gordetzeko aldagaiak ere erazagutuko dira.

Algoritmoa programaren diseinua denez, programazio-lengoaia batera itzuliko da. Programazio-lengoaiak lengoaia artifizial mugatuak dira, eta **exekuzioko kontrol-egitura** zehatz batzuk onartzen dituzte soilik. Beraz, algoritmoetan ere soilik egitura horien pareko diagramak marraztu ahal izango dira. Zazpi kontrol-egitura ditugu: **exekuzio jarraitua** edo lineala; hiru **baldintzapeko adarkatze**-egitura —sinplea, bikoitza, eta anitza—; hiru **egitura errepikakorrak** —errepikatu, den_bitartean, izan_dadin—. Exekuzio jarraitua ezik, beste guztiak baldintza baten menpe daude, bi bide desberdinen artean zein jarraitu behar den erabakitzeko.

Hurrengo ataletan kontrol-egitura hauek azaltzen dira. Egituren erabilera argitzeko asmoz, hirukiaren azalera kalkulatzeko adibidea erabiliko da. Era honetan, zehaztapena eta algoritmoen arteko lotura ere ulertuko da.

Exekuzio jarraitua

Exekuzio-egitura honetan fluxu edo exekuzio-bide bakarra dugu. Aginduak bata bestearen ondoren exekutatzen dira geziari jarraituz. Honako algoritmo honetan exekuzio jarraitua erabiltzen da, hirukien azalera kalkulatzeko.



Ikus daitekeenez, mota berdineko aginduak elementu grafiko bakar batean elkartzen dira (adibidez, sarrera-irteerako aginduak). Horrela eginez gero, aginduen exekuzio-ordena goitik beherakoa da.

Hirukien aldeak izanda, azalera kalkulatzeko zehaztapenetik algoritmoan agertzen diren zenbait elementu ondoriozta daitezke:

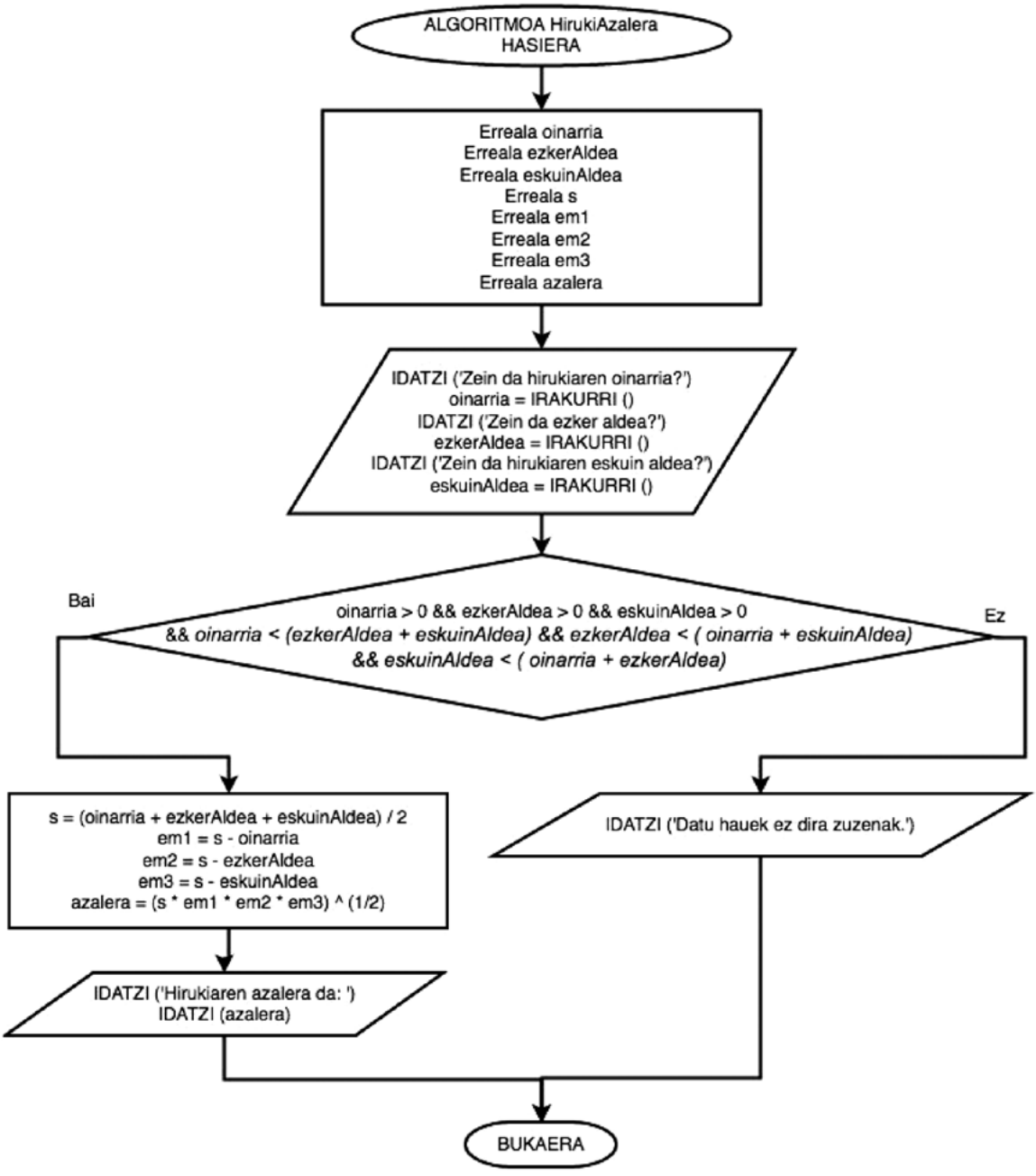
— **Aldagai-erazagupena**. Algoritmo hasieran, zehaztapenak definitzen dituen sarrera eta irteerako datuak erazagutuko dira, izena eta datu mota berdina mantenduz. Gainera, prozesaketan sortzen diren tarteko emaitzak gordetzeko erabiltzen diren beste datuak ere erazagutu behar dira.
— **Sarrerako datuen eskaera**. Prozesaketa egin ahal izateko, sarrerako datu zehatzak beharko dira. Zehaztapenak argitzen du sarrerako datuak *oinarria*, *ezkerAldea* eta *eskuinAldea* direla. Adibideko algoritmo honetan, hirukiaren aldeak erabiltzaileari eskatzen zaizkio.
— **Datu-prozesaketa**. Zehaztapenak ez du zehazten prozesaketa nola egin behar den, baina bai emaitzak eta sarrerako datuak nola erlazionatzen diren, prozesaketa asmatzeko lagungarri izango dena. Adibide honetan, hirukiaren azalera-formulan batuketa-, kenketa-, eta berreketa-eragiketa sinpleak erabiltzen direnez, algoritmoan ia berdin kalkula daiteke. Beste arazo konplexuagoetan, prozesaketa nahasiagoa izan daiteke.
— **Datu-irteera**. Datu-prozesaketaren helburua emaitzak lortzea da. Adibide honetan, zehaztapenak dio emaitza *azalera* dela. Algoritmoan, erabiltzaileari azaltzen zaio datu hori.

Hemen, zehaztapena jartzen dugu, aipaturiko zehaztapen eta algoritmoen arteko erlazio zuzena ikus dadin.

	Aurrebaldintza	**Ondorengo baldintza**
Zer	*oinarria: hirukiaren oinarria metrotan* *ezkerAldea: ezkerreko aldea metrotan* *eskuinAldea: eskuineko aldea metrotan*	*azalera: hirukiaren oinarria metrotan*
Mota	*oinarria: zenbaki erreala* *ezkerAldea: zenbaki erreala* *eskuinAldea: zenbaki erreala*	*azalera: zenbaki erreala*
Mugak	*oinarria > 0 eta ezkerAldea > 0* *eta eskuinAldea > 0*	*azalera > 0*
Erlazioak	*oinarria < (ezkerAldea + eskuinAldea)* *eta ezkerAldea < (oinarria + eskuinAldea)* *eta eskuinAldea < (oinarria + ezkerAldea)*	$azalera = \sqrt{s * (s - oin) * (s - ezA) * (s - esA)}$ *non:* *s = (oin + ezkerAldea + eskuinAldea) / 2* *oin= oinarrria; ezA= ezkerAldea; esA= eskuinAldea*

Baldintzapeko adarkatzea

Egitura honetan bi exekuzio-bide daude. Baldintzaren erantzunaren arabera, exekuzioak *bai adarra* edo *ez adarra* jarraituko du. Hurrengo algoritmoan ikus daitekeenez, aurreko algoritmoari adarkatze bat gehitu zaio sarrerako datuak zuzenak direla baieztatzeko.



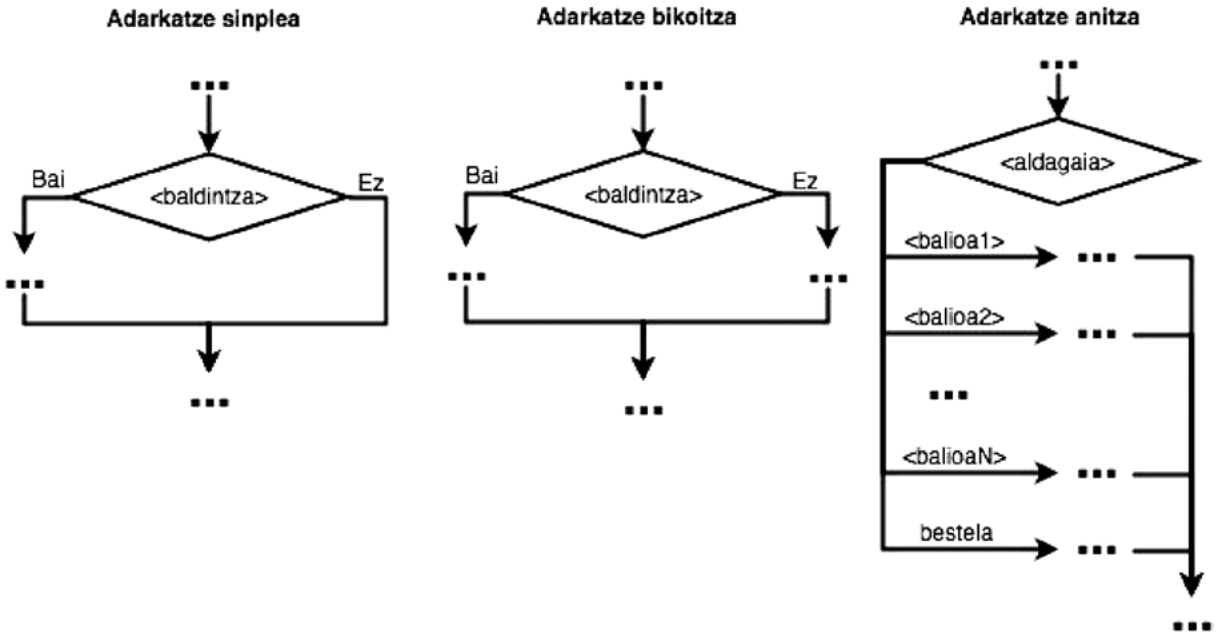
Zenbait algoritmotan, oraingo adibide honetan kasu, erabiltzaileak ematen ditu sarrerako balioak. Baina, agian, nahita ala akats batengatik, ez ditu balio zuzenak ematen; horregatik, komeni da erabiltzaileak emandako datuak baieztatzea. Adarkatzearen erronboan egiten da hori: baldintza

ezezkoa bada (false) *ez* bidea jarraituko da mezu batez datuak desegokiak direla adieraziz; baldintza baiezkoa bada (true) *bai* bidea jarraituko da hirukiaren azalera kalkulatuz.

Gogoratu zehaztapena kontratu moduko bat dela: sarrerako datuek zehaztapenak ezarritako baldintzak betetzen badituzte, programak emaitzei ezarritako baldintzak betearaziko ditu; baina sarrerako datuak ez badira zuzenak, programak ez du zertan kontratua bete behar. Beraz, zehaztapena eta algoritmoaren lotura berri bat dugu:

— **Sarrerako datuen baieztapena**. Zehaztapenaren mugak eta erlazioak ataletan, sarrerako datuei ezarritako baldintzak baieztatu beharko ditu algoritmoak. Behintzat, datu horiek erabiltzaileak ematen baditu. Hirukiaren adibidean, zehaztapenak dio sarrerako datuek positiboak izan behar dutela eta bi aldeen baturak hirugarren aldea baino handiago izan behar duela. Hori betetzen ez bada, deskribatzen den hirukia ez da zuzena, eta ez da bere azalera kalkulatuko.

Hiru adarkatze-egitura ditugu: ***sinplea***, ***bikoitza*** eta ***anitza***. Hemen agertzen dira, hurrenez hurren:

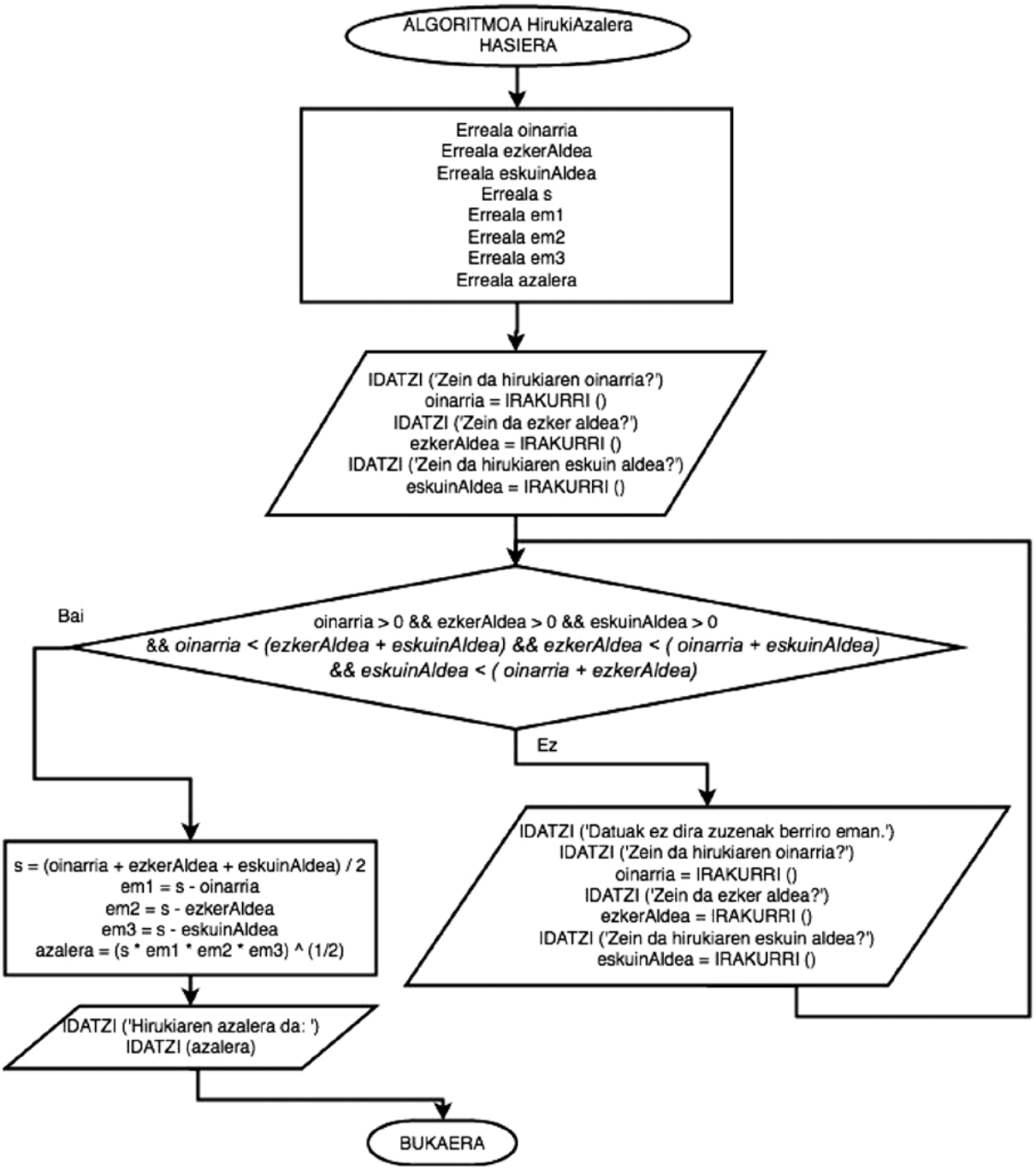


Etenpuntuek (•••) edozein agindu eta kontrol-egitura adierazten dute. Beraz, kontrol-egiturak nahi bezala konbinatu daitezke.

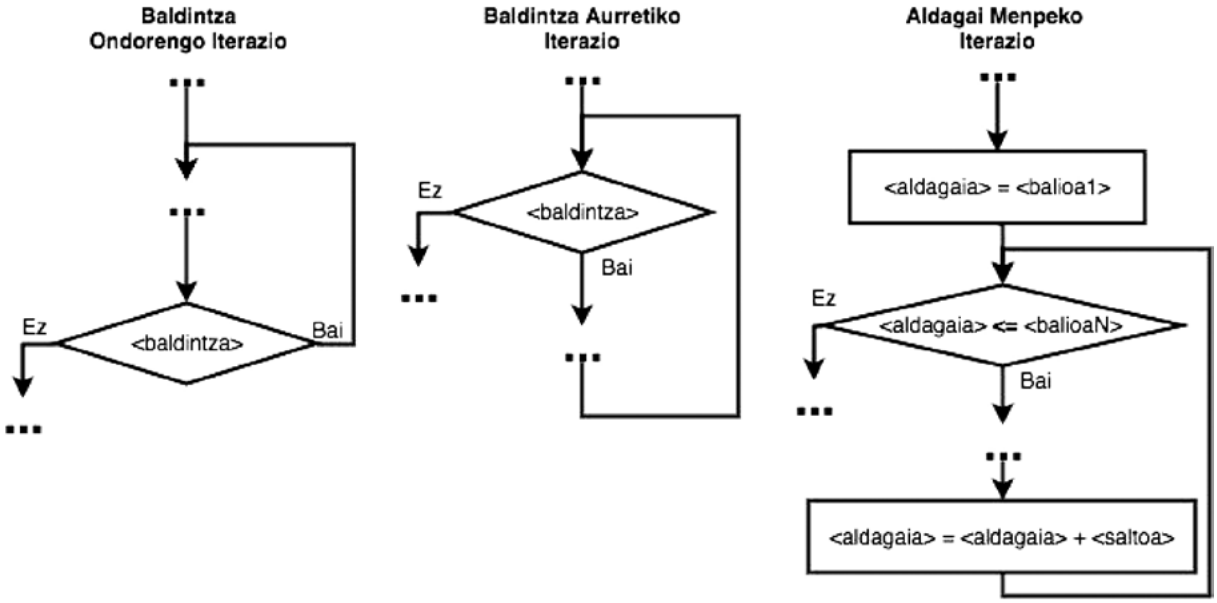
Adarkatze anitzean izan ezik, erronboetan baldintza esplizituak jarriko dira. Baldintzaren baiezko ala ezezko emaitzaren arabera, bai ala ez adarra jarraituko du exekuzioak. Baldintzapeko adarkatze anitzean baldintza inplizitua da. Erronboan agertzen den aldagaiaren balioa adar bakoitzean agertzen den balioekin (<balioa1>, <balioa2>...) konparatuko da ordenan. Aldagaiak duen balioa aurkitzen bada, adar hori jarraituko da. Azken adarrean *bestela* hitza ageri da. Horrek adierazten du, aldagaiaren balioa ez bada aurreko adarretako batean agertzen, azken adar horretatik jarraituko duela exekuzioak. Beraz, adar bakoitzeko baldintza inplizitua da: *<aldagaia> == <balioaX>*.

Exekuzio errepikakorra

Egitura hauetan, errepika daitezkeen exekuzio-begiztak daude. Begizta horiei *iterazio* deritze, eta baldintza baten menpe daude. Adibidez, aurreko algoritmoan sarrerako datuak zuzenak ez badira, programa bukatu egiten da. Egokiagoa izan daiteke, datuak zuzenak ez badira, horiek berriro erabiltzaileari eskatzea. Egitura errepikakor baten bitartez lortzen da hori, hurrengo algoritmoak erakusten duen bezala:



Algoritmoetan hiru egitura errepikakor desberdin erabil daitezke: ***baldintza ondorengoa***, ***baldintza aurretikoa*** eta ***aldagai menpekoa***. Hemen, hurrenez hurren agertzen dira:



Etenpuntuek edozein agindu eta kontrol-egitura adierazten dute. Lehen egiturak baldintza begiztaren bukaeran duenez, begizta barneko aginduak gutxienez behin egikarituko dira. Bigarren egiturak baldintza begizta hasieran du, beraz, gerta daiteke begizta barneko aginduak ez exekutatzea behin ere. Hirugarren kontrol-egitura errepikakorra bigarrenaren berdina da, baina berezitasun bat du: aldagai baten menpe dago. Aldagai batek hainbat balio hartuko ditu, eta balio bakoitzeko begiztan dauden aginduak exekutatuko dira. Era honetan egiten da: baldintza baino lehen aldagaia hasieratu egiten da lehen balioarekin; begiztaren hasierako baldintzak kontrolatzen du ea aldagaiaren balioak <*balioaN*> gainditu duen; ez badu gainditu, iterazioa errepikatzen da; iterazio barneko aginduak exekutatzen badira, azken aginduan aldagaiaren balioa eguneratzen da, duen balioari <saltoa> balioa gehituz. Beraz, aldagaiak balio desberdinak hartuko ditu iterazio bakoitzean.

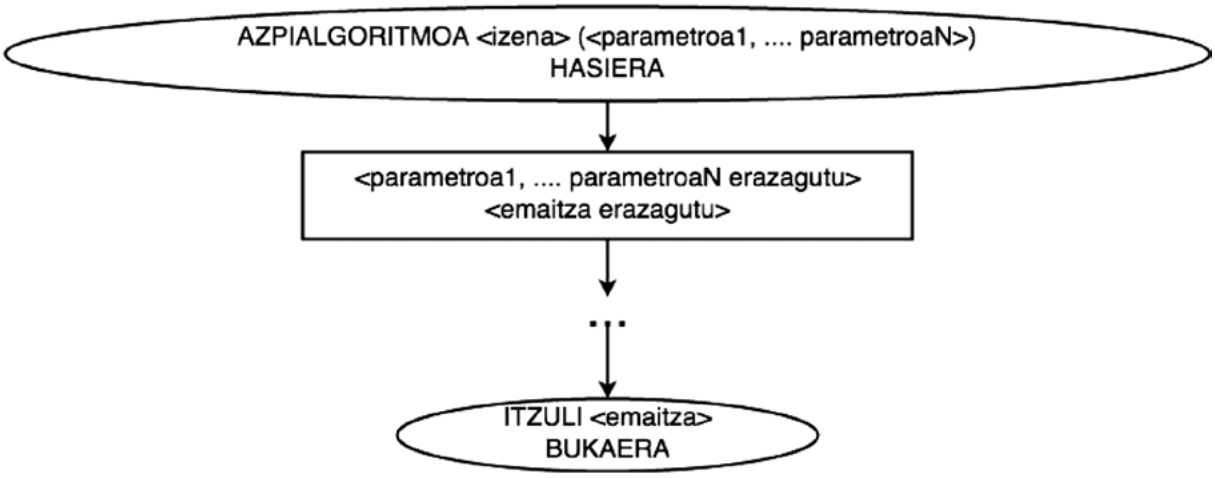
Programazio modularra

Orain arte emandako azalpen eta adibideetan, sarrerako datuak erabiltzaileak ematen ditu, eta emaitzak erabiltzaileari bueltatzen zaizkio. Baina, datu-trukaketa algoritmoen artekoa ere izan daiteke. Zera, **algoritmo nagusi** batek **azpialgoritmo** bati problema zehatz bat ebaztea eskatzea. Algoritmo nagusiak azpialgoritmoari sarrera-datuak emango dizkio, eta horrek emaitza bueltatuko dio ordainetan.

Programazio modularrean, azpiprogramak berrerabili eta konbina daitezkeen moduluak dira. Horren abantaila nagusiak bi dira: alde batetik, **programak berrerabiltzea** denbora aurreztuz; eta, bestetik, **problemak zatitu** beren konplexutasuna txikiagotuz.

Programazio modularra:	«Programazio paradigma bat da, programa bat modulu eta azpi-moduluetan zatitzean datza, irakurgarria eta erabilerraza izateko». *(Wikipedia)*

Azpialgoritmoak era honetan adierazten dira:



Azpialgoritmoetan, parentesi artean **parametro formalak** definitzen dira. Algoritmoetan bezala, parametro formalak erazagutu egiten dira, zein balio mota gordetzen duten adierazteko. Horiek sarrerako datuen balioak jasoko dituzte. Ondoren, parametroen balioak erabiliz emaitza kalkulatzen da. Azkenik, dei egin dion algoritmo nagusiari emaitzaren balioa bueltatuko zaio **ITZULI** aginduaren bitartez. Ikus daitekeenez, ITZULI agindua terminal motako ikur grafiko baten barruan dago (obaloa), hori delako bere portaera: azpialgoritmoaren exekuzioa bukatu, algoritmo nagusiak prozesaketarekin jarrai dezan, bueltatu zaion balioarekin.

Parametro formala:	«Azpirrutina bateko aldagai generiko bat identifikatzen duen sinboloa. Aldagai hau beroni dagokion parametro errealak ordeztuko du programatik deitzen zaionean». *(Euskalterm)*

Algoritmo nagusiak azpialgoritmoari bere izenez deitzen dio, parentesi artean laneko balioak emanez (**parametro erreal**). Azpialgoritmoak bueltatzen duen emaitza aldagai batean gordeko da, esleipena aginduz.

Parametro erreala:	«Prozedura edo funtzio bat abiatzean erabiltzen den benetako aldagaia». *(Euskalterm)*

Problemak zatitu

Divide et impera edo ***banandu eta garaitu*** estrategiaren aplikazioan oinarritzen da. Problema zabalak zailak izaten dira ebazteko, zeren buruan horiek ulertzeak, modelatzeak eta lantzeak esfortzu handia eskatzen baitu. Problema zabal bat problematxo txikiagotan zatitzen badugu errazago ebatziko dugu, nahiz eta problema gehiago ebatzi behar. Problema sinple eta zehatzetan ardazten badugu, problema ebaztea arinago egingo zaigu, zeren bere zehaztapena sinpleagoa ere izango baita.

Adibidez, demagun zenbaki erromatarren kalkulagailu bat programatu nahi dugula. Problema bere osotasunean buruz lantzea nekeza izan daiteke. Nola batzen dira zenbaki erromatarrak? Nola biderkatzen dira? Problema konplexu hori hiru problema sinpleagoren elkarlanarekin ebatz daiteke: erabiltzaileak emandako bi zenbaki erromatarrak hamartar bihurtu; bi zenbaki hamartar horiekin kalkulatu nahi den eragiketa; emaitza hamartarra zenbaki erromatar bihurtu. Horrela ikusita, hiru problema txiki ditugu; hain zuzen ere: 1) zenbaki erromatar bat zenbaki hamartar bihurtu, 2) zenbaki hamartarren kalkulagailua, eta 3) zenbaki hamartar bat zenbaki erromatar bihurtu. Problematxo bakoitza bere aldetik landu daiteke, azkenik algoritmo nagusi batek azpialgoritmo guztiak elkarlanean jarriz. Eskuliburu honen bukaeran lantzen da adibide hori.

Programak berrerabili

Azpialgoritmoak, eta, beraz, azpiprogramak, problema zabal bat ebazteko erabiltzen dira. Askotan, erabilgarriak dira beste problema batean. Adibidez, zenbaki arabiarrak erromatar bihurtzen dituen azpiprogramak erloju erromatar bat egiteko balio dezake; segundoak, minutuak eta orduak zenbaki arabiarrak izango dira, baina erabiltzaileari erakusteko orduan erromatar bihurtzen dira. Azpialgoritmoak berrerabiltzearen abantaila nabaria da: denbora aurrezten dugu, ez baitugu behin eta berriro problema bera ebatzi behar.

Programazio-lengoaiek ere azpiprogramen berrerabilpena ahalbidetzen dute. Ikusi ditugun eragiketa sinpleez gain, eragiketa konplexuagoak ere eskaintzen dituzte. Hau da, programazio-lengoaiek aldez aurretik programatutako azpiprogramak eskaintzen dituzte. Era honetan, ez ditugu berriro programatu behar. Kontuan izan behar da azpiprogramek eginbehar oso zehatza dutela, eta horregatik direla berrerabilgarriak. Adibidez, M programazio-lengoaiak kosinua kalkulatzeko `cosd (<balioa>)` eskaintzen du. Gure kabuz kosinua kalkulatzeko prozedura bat programatu beharrean, zuzenean `cosd` azpiprograma berrerabil dezakegu. Horrelako azpiprograma asko eskaintzen dituzte programazio-lengoaiek. Aurrezten dugun programazio-denbora nabaria da, eta askotan berrerabili izan direnez, egon zitezkeen erroreak azalarazi eta konpondu egingo ziren; beraz, errorerik gabeko azpiprogramak izango dira.

Eskuliburu honen bukaeran dagoen adibide zabalean, LUZERA azpialgoritmoa erabiltzen da. Dimentsio bateko taula baten tamaina jakiteko azpialgoritmoa da. Baina, azpialgoritmoa hori ez da ebazten, zeren M lengoaiak `length` azpiprograma eskaintzen baitigu problema hori ebazteko.

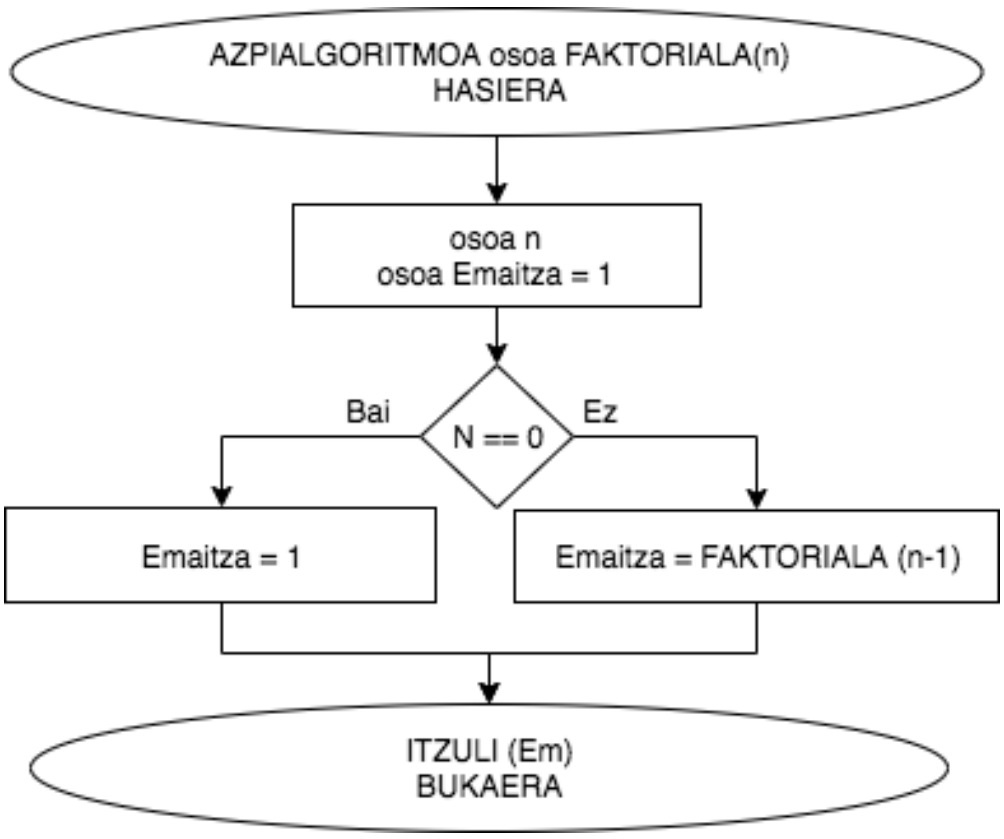
Errekurtsioa

Algoritmoek azpialgoritmoei deitzen diete. Era berean, azpialgoritmoek beste azpialgoritmoei dei diezaiekete. Beren buruari deitzen badiote, **errekurtsio** deritze.

Errekurtsioa:	«Matematikan eta programazioan, errekurtsioa prozesu bat antzeko era batean, bereziki tamaina edo maila txikiagoko problema batera bihurtuz, errepikatzea da, azken kalkulu edo soluzio batera heldu arte». (Wikipedia)

Adibide modura, faktorialaren problema dugu. Hasierako adibideetan azaldu da, eta bere zehaztapena egin da. Laburbilduz, matematikoki ere faktoriala errekurtsioaren bitartez definitzen

da: n! = n * (n – 1)! Oinarrizko kasua 0! da, 1 balioa emaitza duena. Faktoriala azpialgoritmo errekurtsibo batekin ebatz daiteke.



Azpialgoritmoa oso laburra da eta, faktorialaren formulazio matematikoa ulertzen bada, benetan ulergarria. Hala ere, errekurtsioarekin kontu handia izan behar da. Dei errekurtsibo bakoitza exekuzio berri bat da; beraz, exekuzio denbora handitu egingo da eta, aldagai berriak sortzen direnez, ordenagailuko memoria gehiago erabiliko da. Horrek arazo larria sor dezake, dei errekurtsibo asko egiten badira. Faktorialaren kasuan, zenbaki ikaragarri handi batekin gerta daiteke ordenagailua moteltzea, baina oso gutxitan gertatuko da hori. 1.000 zenbakiaren faktorialak 1.000 dei errekurtsibo sortuko ditu.

Aldiz, badira beste problema errekurtsibo batzuk dei kopurua esponentzialki handitzen dutenak. Adibidez, Fibonacciren zenbaki bat kalkulatzeko, aurreko bi Fibonacci zenbakiak kalkulatu behar dira: *Fib (n) = Fib (n – 1) + Fib (n – 2)*. Errekurtsioa oinarrizko 0. eta 1. Fibonacci zenbakietara ailegatutakoan bukatzen da: Fib (1) = 1 eta Fib (0) = 0. Fibonacci zenbaki bakoitza kalkulatzeko bi dei errekurtsibo egiten direnez, guztira dei kopurua 2 ber n inguru izango da. Orduan, 1.000. Fibonaccia kalkulatzeko 2 ber 1.000 dei errekurtsibo egingo dira. Hau da, 10 ber 300 dei baino gehiago. Ordenagailuarentzat lan handiegia da azpiprograma hauek guztiak martxan jartzea, eta ez genuke emaitzarik lortuko.

4. gaia

Ebazpen-prozedura probatu: simulazioak

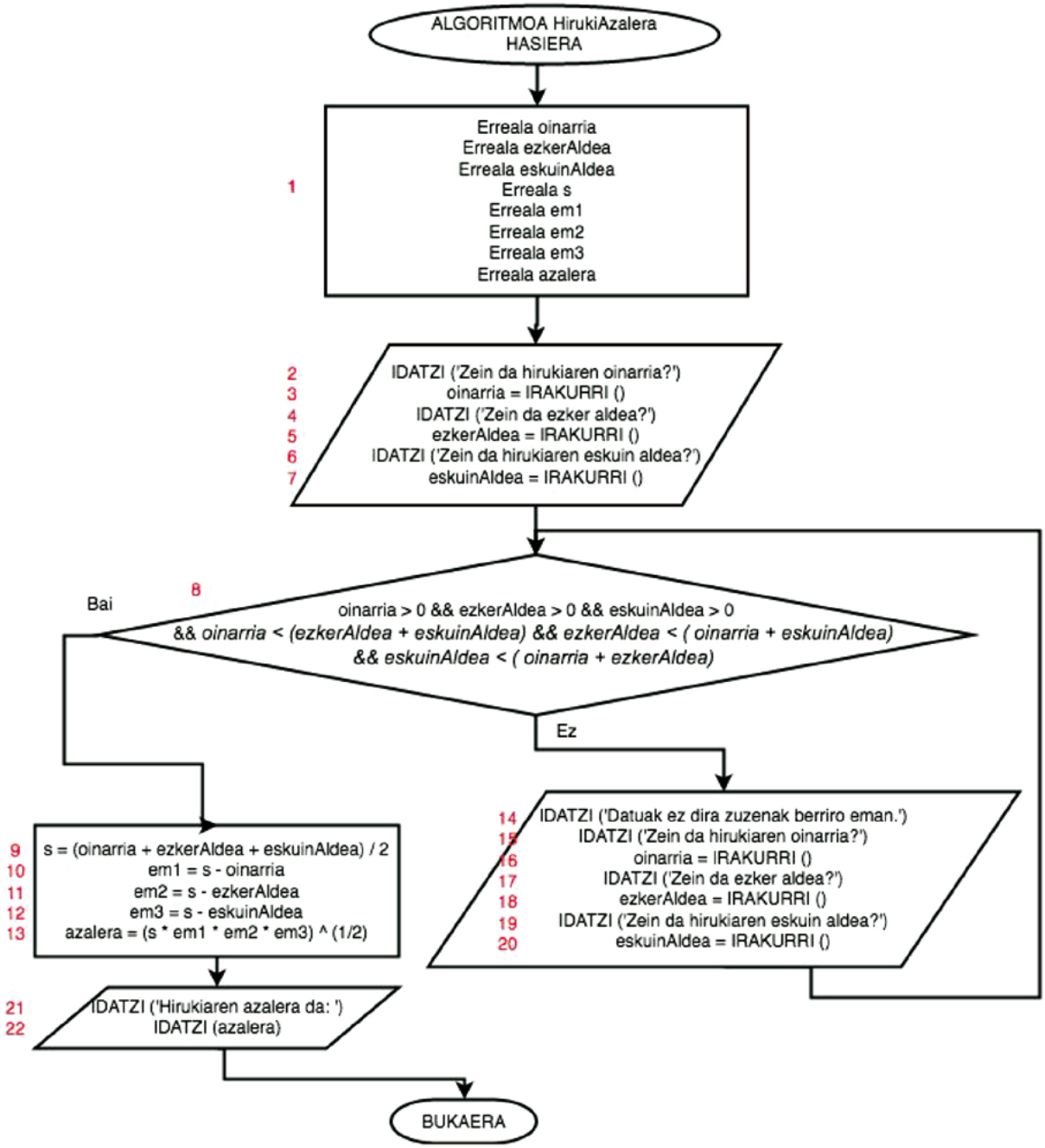
Algoritmoek eta azpialgoritmoek zuzen eta egoki problema bat ebazten dutela baieztatzeko, sarrerako balio zehatzekin exekutatu beharko genituzke. Algoritmoak programaren diseinua dira, ez dira berez exekutagarriak. Horregatik, exekuzioaren simulazioa egiten dugu. Sarrerako balio batzuk jasota, algoritmoaren exekuzio-fluxua nolakoa den eta aldagaien balioak nola aldatzen diren erakutsiko digu simulazioak. Horren helburua da prozedurak emaitza zuzena lortzen duela baieztatzea, eta emaitza zuzena ez bada akatsak non dauden atzematea.

Simulazioetan, zehaztapenean egiten den bezala, kasu orokorretan eta berezietan pentsatzen da. Aldez aurretik, emaitza edo algoritmoaren portaera zuzena zein den jakin behar dugu. Simulazioak erakusten duena ez badator bat espero dugunarekin, akats bat dago, eta non gertatu den aurkitu beharko da. Hirukiaren azalera kalkulatzeko, algoritmoan honako kasu hauek proba ditzakegu:

1. Sarrerako balio desegokiak:
   1.1. Alde neurri bat negatiboa. Adibidez, *oinarria* –1, *ezkerAldea* –2 eta *eskuinAldea* –3 badira, algoritmoak datuak berriro eskatu beharko lituzke.
   1.2. Bi aldeen batura hirugarrena baino txikiagoa izatea. Adibidez, *oinarria* 1, *ezkerAldea* 2 eta *eskuinAldea* 10 badira, orduan datuak berriro eskatu beharko lituzke algoritmoak.
2. Sarrerako balio egokiak. Adibidez, *oinarria* 5, *ezkerAldea* 6 eta *eskuinAldea* 7 badira, emaitza zuzenak 14’7 izan behar du.

Simulazioan ikusiko dugu momentu bakoitzean zein agindu exekutatzen den eta aldagaien balioak nola aldatzen diren. Horretarako, probatu nahi dugun algoritmoaren aginduak zenbakien bitartez identifikatuko ditugu. Hemen erakusten da hirukiaren algoritmoa izendatuta.

Zenbakien ordenak ez du axola, kontua da zenbaki bakoitzak agindu eta baldintza bakarra identifikatzea. Simulazioa laburtzeko, aldagai-erazagupen osoa 1 zenbakiaz identifikatu da, gauza gehiegi gertatzen ez delako bertan. 8 aginduak erronboaren baldintza identifikatzen du. Baldintza bakarra da, nahiz eta hiru lerrotan idatzi algoritmoa konpaktuago egiteko. Baldintzetan ez dira aldagaien balioak aldatzen, baina exekuzioak hartzen duen bidea erabakitzen da.



Simulazio-taula batean idatziko da, non errenkada bakoitzak exekuzioaren momentu zehatz batean aldagaien balioak erakutsiko baititu. Lehen zutabean zein agindu exekutatzen ari den identifikatuko da; ondorengo zutabeetan aldagai bakoitzaren balioa, eta, azkenengoan, erabiltzaileari erakusten zaizkion mezuak pantailaz. Exekuzio-simulazio bakoitzak taula bat sortzen du. Sarrerako datuak zuzenak ez direnean hirukiaren algoritmoak berriro eskatzen ditu; beraz, lehen zehaztu ditugun probatu beharreko kasuek taula bakarra sortuko dute.

Hemen aurkezten den simulazio-taulan hiru kasu desberdin probatu dira. Hiruretan algoritmoak egoki funtzionatzen du. Lehenengo kasuan, alde negatiboak jasotzen ditu, eta ez ditu onar-

tzen. Berriro eskatzen ditu datuak. Bigarren kasuan, balioak ere ez ditu onartzen, aldeen balioek bete beharreko baldintzak betetzen ez dituztelako. Taulari erreparatuz, egitura errepikakorraren lana ikus daiteke: 8 eta 20 arteko aginduak bi aldiz errepikatzen dira. 8. aginduan egituraren baldintza ebaluatzen da momentuko balioekin; baldintza betetzen ez bada errepikatu egiten da. Berriro datuak eskatzen direnean, hirugarren kasuko datu zuzenak ematen dira. Jada 8. aginduko baldintzak *bai* balioa ematen du, eta egitura errepikakorretik irteten da exekuzioa. Datu zuzenak direnez, hirukiaren azalera kalkulatzen da, eta erabiltzaileari espero zen emaitza erakusten zaio.

Azpialgoritmoak berdin proba daitezke simulazio-taulen bitartez. Kasu bereziak eta orokorrak identifikatu: sarrerako balioak aukeratu eta espero den emaitza kalkulatu. Sarrerako balioak parametro formalek dituzten balioak izango dira azpialgoritmoa martxan jartzen denean.

	#	oinarria	ezkerAldea	eskuinAldea	s	em1	em2	em3	azalera	Pantaila
	1	?	?	?	?	?	?	?	?	
	2	?	?	?	?	?	?	?	?	'Zein da ...'
	3	-1	?	?	?	?	?	?	?	
	4	-1	?	?	?	?	?	?	?	'Zein da ...'
	5	-1	-2	?	?	?	?	?	?	
	6	-1	-2	?	?	?	?	?	?	'Zein da ...'
	7	-1	-2	-3	?	?	?	?	?	
ez	**8**	-1	-2	-3	?	?	?	?	?	
	14	-1	-2	-3	?	?	?	?	?	'Datuak ez ...'
	15	-1	-2	-3	?	?	?	?	?	'Zein da ...'
	16	1	-2	-3	?	?	?	?	?	
	17	1	-2	-3	?	?	?	?	?	'Zein da ...'
	18	1	2	-3	?	?	?	?	?	
	19	1	2	-3	?	?	?	?	?	'Zein da ...'
	20	1	2	10	?	?	?	?	?	
ez	**8**	1	2	10	?	?	?	?	?	
	14	1	2	10	?	?	?	?	?	'Datuak ez ...'
	15	1	2	10	?	?	?	?	?	'Zein da ...'
	16	5	2	10	?	?	?	?	?	
	17	5	2	10	?	?	?	?	?	'Zein da ...'
	18	5	6	10	?	?	?	?	?	
	19	5	6	10	?	?	?	?	?	'Zein da ...'
	20	5	6	7	?	?	?	?	?	
bai	**8**	5	6	7	?	?	?	?	?	
	9	5	6	7	9	?	?	?	?	
	10	5	6	7	9	4	?	?	?	
	11	5	6	7	9	4	3	?	?	
	12	5	6	7	9	4	3	2	?	
	13	5	6	7	9	4	3	2	14.7	
	21	5	6	7	9	4	3	2	14.7	'Hirukiaren...'
	22	5	6	7	9	4	3	2	14.7	14.7

5. gaia

Algoritmoa itzuli: programa

Algoritmoak problemaren ebazpena lortzeko jarraitu beharreko prozedura zehazten du. Beraz, problema ebatzi da. Programa sortzeko dugun algoritmoa programazio-lengoaiara itzuli beharko da. Horregatik, algoritmoa programaren diseinua da. Programa horiek algoritmoen ezaugarri berdinak izango dituzte:

— Zehatzak dira.
— Pausoen arteko ordena ezaguna da.
— Beti amaitzen dute.
— Sarrerako datu berdinekin programa bi aldiz exekutatuz, emaitza berdina lortuko da.

Eskuliburu honetan aukeratu den programazio-lengoaia M lengoaia da. M lengoaia Mathworks enpresak (https://es.mathworks.com/) asmatutako programazio-lengoaia da. M lengoaian idatzitako programak ez dira zuzenean exekutagarriak, Garapenerako Ingurune Bateratu batek egikaritzen ditu. Hau da, programak ingelesean oinarritutako lengoaia batean idatzita daude hitzez hitz, eta M luzapena duten fitxategietan gordeta. Garapenerako Inguruneek fitxategia ireki eta M lengoaiako programa interpretatzen dute dute. Hau da, aginduak banan-banan irakurri eta exekutatzen dituzte. Garapenerako Ingurune ezagunak MATLAB (https://es.mathworks.com/products.html?s_tid=gn_ps) eta debaldeko alternatiba Octave GNU (https://www.gnu.org/software/octave/) dira. MATLAB, MATrix LABoratory hitzen laburpena da, M lengoaia matrizeekin lan egiteko pentsatuta baitago. Matrizea (ala taula) oinarrizko datu mota du, eta era eraginkorrean erabiltzen du. MATLAB-ek eta Octave GNUk zati nagusi hauek erakusten dituzte:

1. Fitxategi-kudeaketa. Programa-fitxategiak menuko aukeren bitartez kudeatzen dira: ireki, itxi, berria, etab.
2. Editorea. Programa berri bat idazteko erabilgarriak diren laguntzak: bilatu eta aldatu, sintaxi-koloreztaketa automatikoa, etab.
3. Agindu-leihoa. Erabiltzaileak zuzenean aginduak helarazi diezazkioke inguruneari exekuta ditzan.
4. Langunea. Exekuzioan erabilgarri dauden datuak gordetzen ditu.

M programazio-lengoaiak programazio inperatiboa, egituratua eta modularra ahalbidetzen ditu. Itzulpena egiteko algoritmoetan ager daitezkeen elementuak M lengoaian nola adierazten diren jakin behar da. Eskuliburu honek programazioaren oinarriak erakusteko balio du, ez M progra-

mazio-lengoaia sakonean ikasteko. Algoritmoak itzultzeko beharrezkoak diren aginduak azalduko dira soilik.

Fluxu-diagrametan, ikur grafiko eta hitzekin adierazten dira algoritmoak. Aldiz, M lengoaia textuala da. Beraz, hitz erreserbatu eta aginduen bitartez adierazi behar da algoritmoek adierazten duten gauza bera.

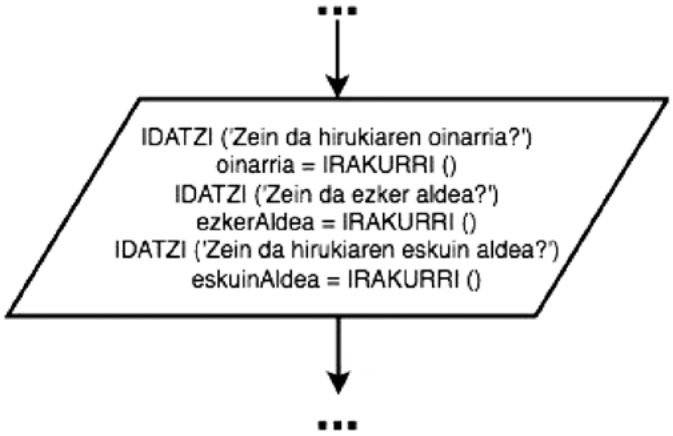
Programazio inperatiboa

Algoritmoen esleipenetan eta baldintzetan jarri dena berdin adierazten da M lengoaian. Alegia, balio konstanteak, aldagai-izenak, taulak sortu eta atzitzea, eragiketa matematikoak, eragiketa erlazionalak eta lokarri logikoak. Beraz, algoritmoetan jartzen duena programan kopia dezakegu.

Aldatzen den gauza bakarra Sarrera/Irteerako aginduak adierazteko era da. IRAKURRI agindua M lengoaian *input* da, eta IDATZI agindua *disp*. M lengoaiako aginduak eta azpiprogramak zehazki nola erabiltzen diren jakiteko MATLAB-eko agindu-leihoan *help <aginduizena>* exekutatu.

disp aginduak parentesi artean ematen zaiona pantailaratzen du. *input* zain egoten da erabiltzaileak teklatuz balio bat eman arte. Ondoren *input*ak bueltatzen du, eta esleipenaren bitartez aldagai batean gordetzen da. Aurreko adibidean ikus daiteke azken *input*ak mezua ere pantailaratzeko aukera duela. Beraz, IDATZI eta IRAKURRI aginduen lana egin dezake.

Fluxu-diagrama: Sarrera/Irteera aginduak



M programazio-lengoaia: *input* eta *disp*

```
…
disp ('Zein da hirukiaren oinarria?')
oinarria = input ('');
disp ('Zein da ezker aldea?')
ezkerAldea = input ('');
eskuinAldea = input ('Zein da eskuin aldea?');
…
```

Programazio egituratua

Fluxu-diagraman erronbo bat agertzen denean kontrol-egitura bat dago. Erronboetan exekuzioak zein adarretatik joko duen erabakitzeko baldintza bat dago. Esan bezala, eragiketa erlazionalak eta lokarri logikoak berdinak dira algoritmoetan eta M lengoaian. Beraz, kontrol-egituren baldintzak berdin idazten dira.

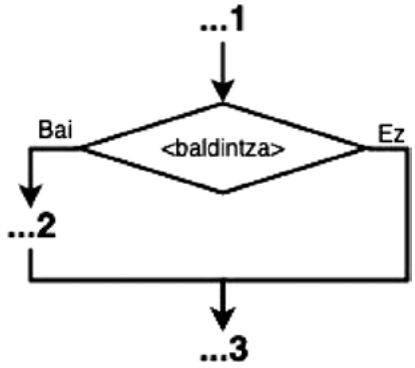
Exekuzio jarraitua

Algoritmoetan exekuzioaren ordena geziari jarraituz da, ez dago adarkatzerik ez errepikapenik. M lengoaian exekuzioa fitxategiaren hasieratik bukaera artekoa da. Algoritmoaren ordena berdina jarraituz idazten dira aginduak programan. Aurreko adibidean ikus daiteke sarrerako eta irteerako aginduen ordena mantentzen dela.

Baldintzapeko adarkatzea

Hiru adarkatze-egitura ditugu: ***sinplea***, ***bikoitza*** eta ***anitza***. Hemen agertzen dira hurrenez hurren beren itzulpenak M lengoaiara.

Fluxu-diagrama: baldintzapeko adarkatze sinplea

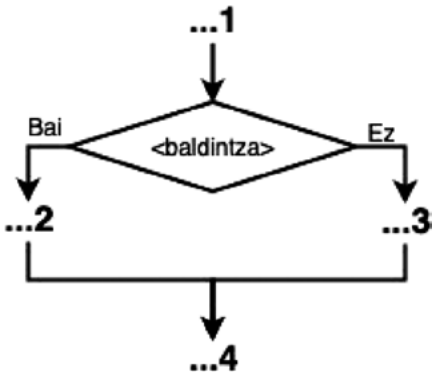


M programazio-lengoaia: `if ... end`

```
…1
if ( <baldintza> )
     …2
end
…3
```

M lengoaian *if* eta *end* hitzen artean dagoena exekutatuko da, baldin eta *<baldintza>* baiezkoa bada (*bai* bidean dauden *…2* aginduak).

Fluxu-diagrama: baldintzapeko adarkatze bikoitza

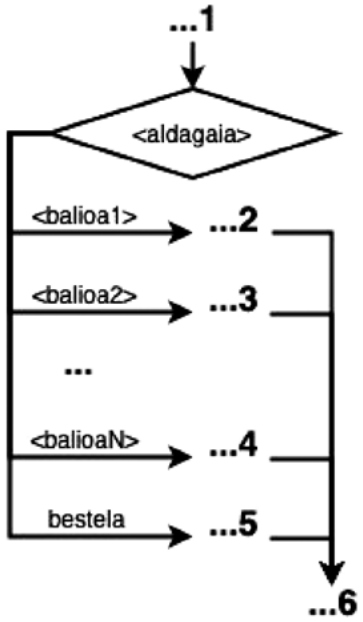


M programazio-lengoaia: *if … else … end*

```
…1
if ( <baldintza> )
     …2
else
     …3
end
…4
```

Adarkatze sinplean bezala, *<baldintza>* baiezkoa bada *if* eta *else* hitzen artean dagoena exekutatuko da (*bai* bidean dauden *…2* aginduak), baina ezezkoa bada *else* eta *end* artean dagoena exekutatuko da (*ez* bidean dauden *…3* aginduak).

Fluxu-diagrama: baldintzapeko adarkatze anitza



M programazio-lengoaia: *switch … case … otherwise … end*

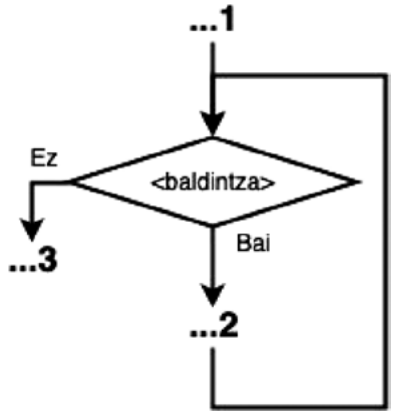
```
…1
switch ( <aldagaia> )
case <balioa1>
…2
case <balioa2>
…3
…
case <balioaN>
…4
otherwise
…5
end
…6
```

switch egituraren hasieran agertzen den aldagaiaren balioak zehazten du exekutatzen dena. *case* batek aldagaiaren balioa badu, bertako aginduak exekutatuko dira (*…2*, *…3*, ala *…4*). Ez badago aldagaiaren balioa duen *case*rik, orduan *otherwise* parteko aginduak exekutatuko dira (*…5*).

Exekuzio errepikakorra

Hiru egitura errepikakor ditugu: ***baldintza aurretik***, ***baldintza ondoren*** eta ***balio-segida***. Hemen, hurrenez hurren agertzen dira.

Fluxu-diagrama: errepikakorra baldintza aurretik

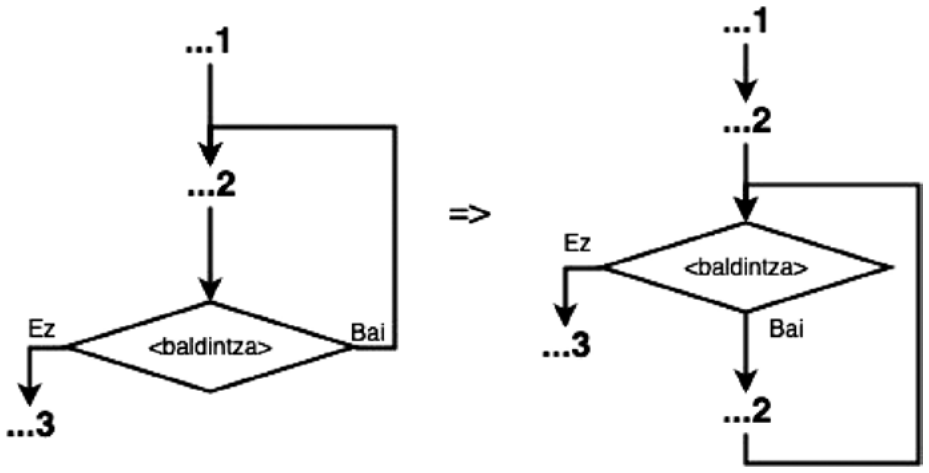


M programazio-lengoaia: `while ... end`

```
…1
while ( <baldintza> )
…2
end
…3
```

Baldintza betetzen den bitartean, *while* eta *end* hitzen arteko aginduak exekutatuko dira (*...2*).

Fluxu-diagrama: errepikakorra baldintza ondoren

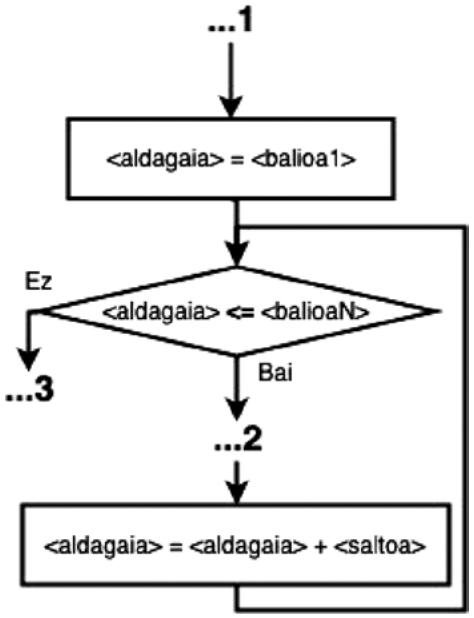


M programazio-lengoaia: … *while* … *end*

```
…1
…2
while ( <baldintza> )
…2
end
…3
```

M lengoaian ez dago baldintza ondoren duen egitura errepikakorrik. Hortaz, egitura hori baldintza duen egitura errepikakor bihurtu behar dugu. Portaera berdina izateko errepikatzen diren aginduak (*…2*) egitura errepikakorra baino lehen bikoiztu behar ditugu. Orain bai, pareko egitura M lengoaian da `while`.

Fluxu-diagrama: errepikakorra balio-segida



M programazio-lengoaia: `for` … `end`

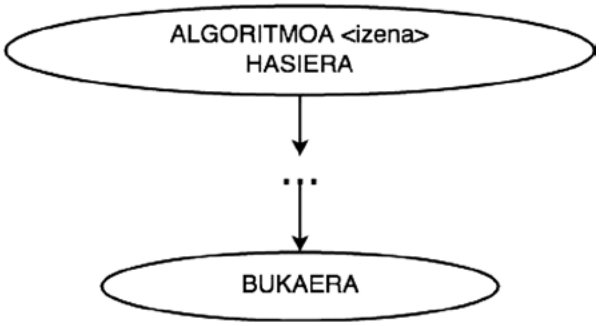
```
…1
for ( <aldagaia> = <balioa1> : <saltoa> :
<balioaN>)
…2
end
…3
```

Algoritmoan duguna `while` egitura batekin adieraz daiteke M lengoaian. Baina hemen azaltzen den aldagai bati zenbaki-segida bateko balioak ematea ohikoa izaten da programazioan (taula bateko posizioak korritzeko adibidez), eta, horregatik, programazio-lengoaietan egitura bat dago hori adierazteko. `for` egitura da M lengoaian, eta bere goiburukoan agertzen dira segida definitzeko elementu guztiak: aldagaia, hasierako balioa, eguneraketa eta muga zehazten duen balioa.

Programazio modularra

Algoritmoak programa nagusiak izango dira M lengoaian. Horien barnean aginduak egongo dira; besteak beste, azpiprogramei deiak.

Fluxu-diagrama: algoritmoa

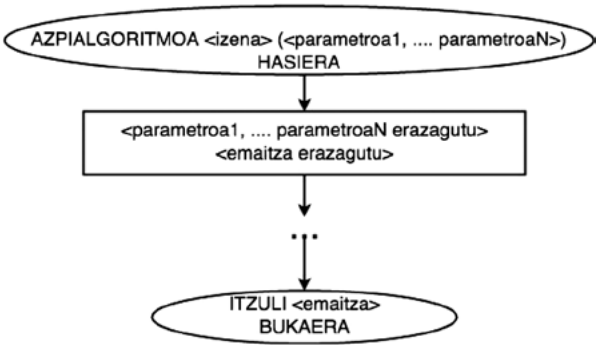


M programazio-lengoaia: *function* ... *end*

```
function <izena>
…
end
```

function hitz erreserbatuak programaren hasiera zehazten du; bere bikotea den *end* hitz erreserbatuak programa bukatzen du. Bien artean aginduak eta kontrol-egiturak egongo dira.

Fluxu-diagrama: azpialgoritmoa



M programazio-lengoaia: *function* ... *end*

```
function <emaitza> = <izena> (<parametroa1>,…, <parametroaN>)
…
end
```

M lengoaian programak eta azpiprogramak *function* hitzarekin adierazten dira. Desberdintasuna da azpiprogramek parametro formalak eta bueltatu beharreko aldagaiak dituztela goiburukoan. M lengoaian ez dago ***ITZULI*** pareko agindurik. Hala eta guztiz ere, azpiprogramaren portaera algoritmoaren berdina da. Azpiprograma bukatzen denean, <*emaitza*> aldagaiak duen balioa bueltatzen zaio programa deitzaileari.

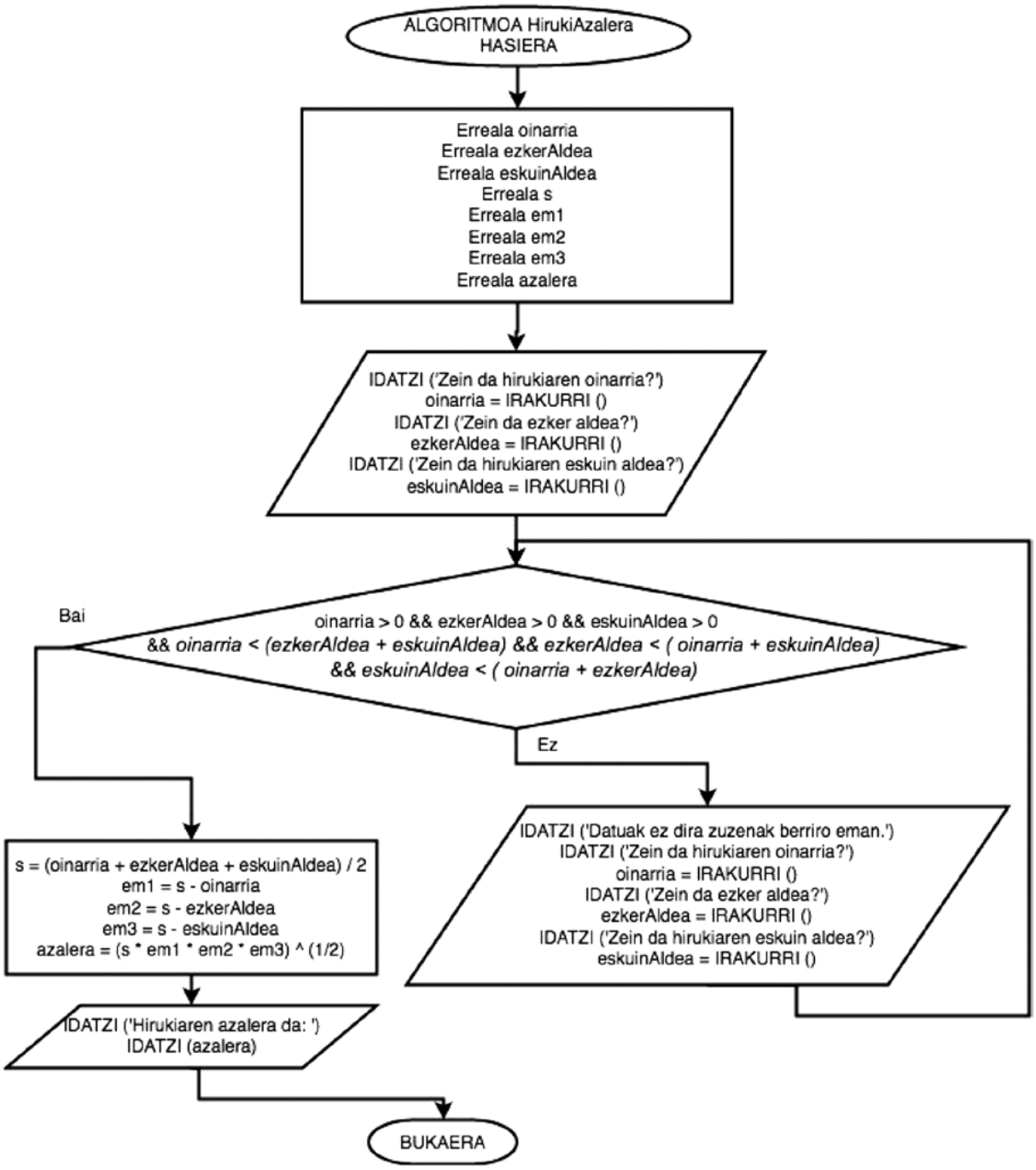
Nabarmentzekoa da M lengoaiak ez dituela aldagaiak erazagutzen. Balio bat esleitzen zaion lehenengo aldian sortzen da aldagaia.

6. gaia

Programa araztu: exekuzioak

Algoritmoa M lengoaiara itzuli ondoren, MATLAB-en edo Octave-n exekuta daiteke. Kasu bereziak eta orokorrak probatuko genituzke. Emaitzak zuzenak ez badira, errorea non dagoen aurkitu beharko da. Hori arazketaren bitartez egin daiteke. Hau da, exekuzioa pausoz pauso egin aldagaien balioak zein diren ikusiz. Simulazio-taula modukoa da, baina ordenagailuan automatikoki. Errorea itzulpen-akats bat izan daiteke, ala prozedura zuzena ez dela diseinatu ebazpen fasean. Horren arabera, programa editatu ala algoritmoa aldatu beharko da.

Programa baten exekuzioa eta arazketa probatzeko hirukiaren algoritmoa itzuliko da, irakurleak MATLAB-en edo Octave-n exekuta dezan. Horretarako, MATLAB-en edo Octave-n .m luzapena duen fitxategi bat sortuko genuke, eta bertan algoritmoaren itzulpena idatziko genuke.



Aurreko atalean erakutsitako itzulpenei jarraituz, honako programa hau izango genuke.

HirukiAzalera.m

```
1   function HirukiAzalera
2 -     oinarria = input ('Zein da hirukiaren oinarria?');
3 -     ezkerAldea = input ('Zein da ezker aldea?');
4 -     eskuinAldea = input ('Zein da hirukiaren eskuin aldea?');
5 -     while ~(oinarria > 0 && ezkerAldea > 0 && eskuinAldea > 0  && ezkerAldea < ( oinarria + eskuinAldea) ...
6               && oinarria < (ezkerAldea + eskuinAldea) && eskuinAldea < ( oinarria + ezkerAldea))
7           oinarria = input ('Zein da hirukiaren oinarria?');
8           ezkerAldea = input ('Zein da ezker aldea?');
9           eskuinAldea = input ('Zein da hirukiaren eskuin aldea?');
10      end  %while bukaera
11      s = (oinarria + ezkerAldea + eskuinAldea) / 2;
12      em1 = s - oinarria;
13      em2 = s - ezkerAldea;
14      em3 = s - eskuinAldea;
15      azalera = (s * em1 * em2 * em3) ^ (1/2);
16      disp ('Hirukiaren azalera da: ')
17      disp (azalera)
18  end %function bukaera
```

Command Window

```
>> HirukiAzalera
Zein da hirukiaren oinarria? 5
Zein da ezker aldea? 6
Zein da hirukiaren eskuin aldea? 7
Hirukiaren azalera da:
   14.6969
```

Nabarmentzekoa da `while` egitura errepikakorraren baldintza ukatuta dagoela. Algoritmoan erreparatuz ikus daiteke begizta errepikakorra ezezko bidea dela; hau da, baldintza betetzen ez denean errepikatzen da. M lengoaian `while` egiturak errepikatzen du baldintza betetzen denean; horregatik, ukapena agertzen da aurretik.

Aurreko irudian *Command Window* leihoan agertzen da programaren exekuzioa. *HirukiAzalera* izenaren bitartez programa martxan jartzen da. Horretarako, .m fitxategiak programaren izena izan behar du.

7. gaia

Adibide orokorra

Eskuliburu honetan aipatutakoak elkarrekin ikusteko, adibide baten pauso guztiak ikusiko dira atal honetan. Problemaren enuntziatua sinplea da: «Zenbaki erromatarren kalkulagailu bat nahi da». Enuntziatua sinplea izan arren, ebazpena konplexua izan daiteke. Gainera, enuntziatuak adierazi gabekoak erabaki beharko dira. Adibidez, zein eragiketa onartuko ditu kalkulagailuak? Beraz, lehenengo pausoan problema ondo ulertu eta modelizatu beharko da.

Modelizazioa

Zenbaki erromatarrak zenbaki oso eta positiboen kodeketa-sistema berezi bat dira. Oinarrizko karaktere batzuen konbinazioen bitartez zenbakiak adierazten dira. Oinarrizko karaktereak eta beren zenbaki-balioak hauek dira, unitate-mailatan sailkatuta:

— Batekoak:

- ‘I’: bat balio du.
- ‘V’: bost balio du.

— Hamarrekoak:

- ‘X’: hamar balio du.
- ‘L’: berrogeita hamar balio du.

— Ehunekoak:

- ‘C’: ehun balio du.
- ‘D’: bostehun balio du.

— Milakoak:

- ‘M’: mila balio du.

Karaktereak konbinatu egiten dira zenbaki baten balioa osatzeko, karaktereen balioa batuz zenbakiaren balioa kalkulatzen baita. Karaktere balioaren arabera handienetik txikienera ordenatuta ageri dira, eta errepika daitezke behar izanez gero. Adibidez, DCXXIII zenbaki erromatarra 623 arabiar zenbakia da. Erromatar zenbakera agregazio-sistema inkremental bat da. Zera esan

nahi du horrek: balio handiagoa duen karaktere batez adieraz badaiteke karaktere multzo batek batzen duena, orduan ordezkatu egingo du. Adibidez, 'V' karaktereak bost 'I' ordezkatzen ditu, eta 'X' karaktereak bi 'V' ordezkatzen ditu. Izan ere, 623 zenbakia adierazteko, 623 'I' karaktere bat bestearen ondoren jartzea ez da oso eraginkorra. Badu berezitasun txiki bat agregazio-sistema honek: lau karaktereen multzoak ez dira onartzen; horien ordez, karaktereen ordena aldatuz kenketa adierazten da. Adibidez, lau 'I' multzoaren ordez 'IV' jartzen da, ulertuz bost balioari bat kendu behar zaiola. Onartzen ez diren lau karaktereen multzoak honako hauek besterik ez dira:

— 'IIII' ordez 'IV', lau balioa adieraziz.
— 'VIIII' ordez 'VIV' izango genuke; baina 'X' karaktereak bi 'V' ordezkatzen dituenez, 'IX' izango genuke bederatzi balioa adierazteko (10 – 1). Ez da 'XI' sortzen, horrek hamaika adierazten baitu (10 + 1).
— 'XXXX' ordez 'XL', berrogei balioa adieraziz.
— 'LXXXX' ordez 'LXL' izango genuke; baina 'C' karaktereak bi 'L' ordezkatzen dituenez, 'XC' izango genuke laurogeita hamar balioa adierazteko.
— 'CCCC' ordez 'CD', laurehun balioa adieraziz.
— 'DCCCC' ordez 'DCD' izango genuke; baina 'M' karaktereak bi 'D' ordezkatzen dituenez, 'CM' izango genuke bederatziehun balioa adierazteko.

Beraz, balioaren araberako ordenazioa handitik txikienera aldatu egiten da kasu hauetan, kenketa adieraziz. Baina kenketa zerrendatutako sei kasu horietan soilik agertzen da. Norbaitek pentsa dezake 'IM' 1999 balioa adierazteko era egokia dela. Baina ez, erromatar zenbakera inkrementala da. 1999 balioaren aurretik 1998 balioa dugu, erromatarrez adierazita: 'MCMXCVIII'. Bat gehitzen badiogu, oker adierazita, 'MCMXCVIIII' izango litzateke. Aurreko zerrendan aipatu denez, 'MCMXCIX' izango litzateke konbinazio zuzena. Beraz, 1999 balioa zuzen adierazita 'MCMXCIX' da. Nabarmentzekoa da, baita ere, zenbaki erromatarrak irakurtzeko errazak direla era horretan, ezkerretik eskuinera lehenengo milakoak agertzen direlako, ondoren ehunekoak, gero hamarrekoak, eta azkenik batekoak.

Azalpen horiek guztiak eta gero, erabaki dezakegu nolakoa izango den kalkulagailu erromatarra:

— Eragiketa bitar hauek onartuko ditu: kenketa ('–'), batuketa ('+') eta biderketa ('*').
— Sarrerako eta irteerako zenbaki erromatarren balioak zero baino handiagoak eta 4.000 baino txikiagoak izango dira. 4.000 baliotik gorako zenbakiak erromatarrez adieraz daitezke, baina adierazpena zaildu egiten da. Sinplifikatze aldera, muga hori ezartzen da.
— Kenketaren bigarren eragigaiaren balioa lehen eragigaiaren balioa baino txikiagoa izan behar da.

Emaitza zenbaki erromatar bat izango da. Sarrerako datuak adierazpen matematiko bakar baten modura adieraztea erabaki da. Adibidez, 'V + IX'. Hau da, adierazpen matematikoaren itxura izango du: *<zenbakiErromatar1> <eragiketa> <zenbakiErromatar2>*.

Zehaztapena

Arazoa ulertu eta modelizatu ondoren, zehaztapena definituko dugu. Hau izan daiteke:

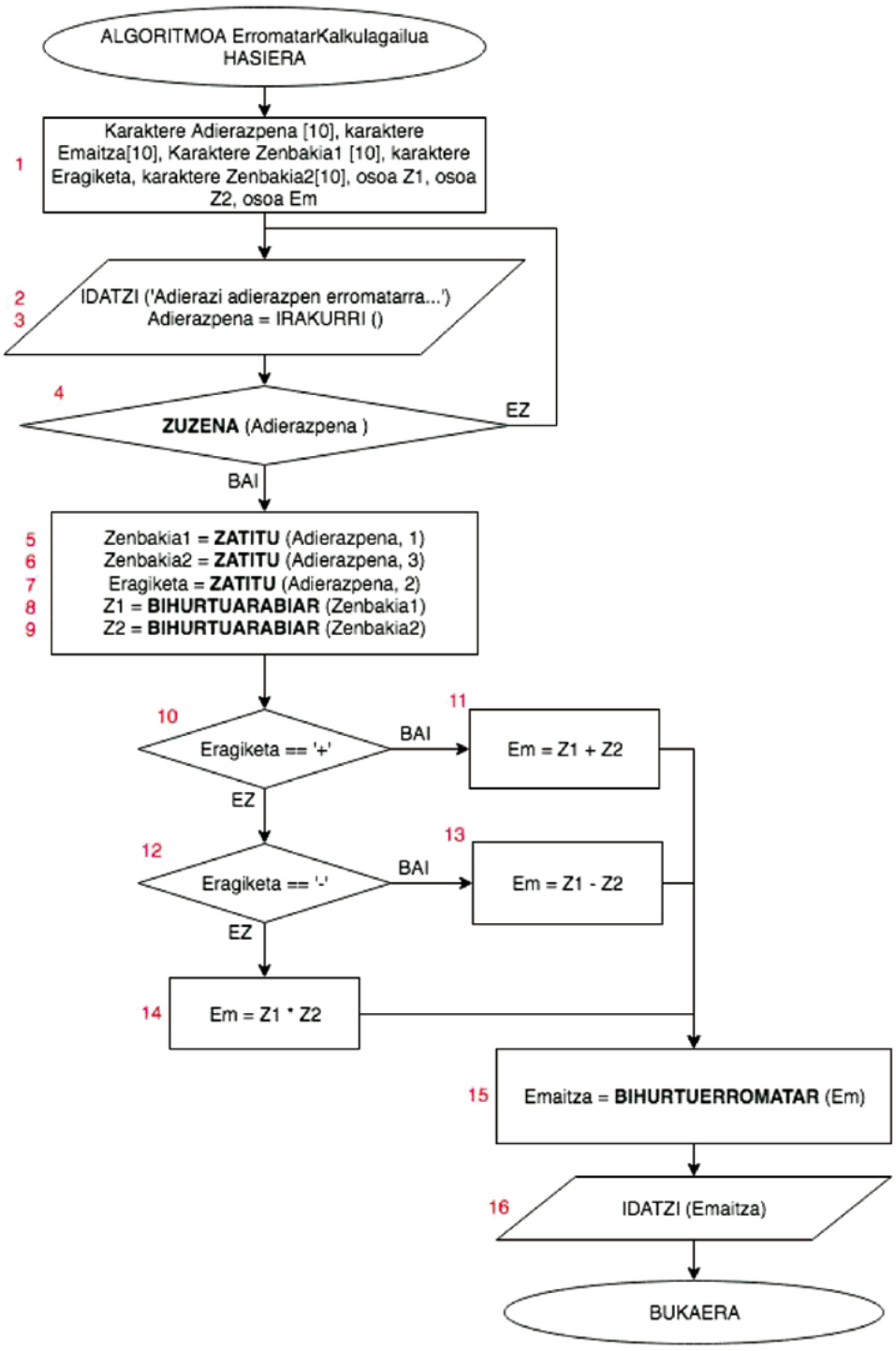
	Aurrebaldintza	Ondorengo baldintza
Zer	**Adierazpena**: kalkulatu nahi den adierazpen matematikoa zenbaki erromatarrekin. Adibidez '*V + IX*'.	***Emaitza:*** adierazpenaren emaitza zenbaki erromatarretan adierazia. Adibidez *'XIV'*.
Mota	***Adierazpena***: karaktere-taula (tamaina ezin da jakin aldez aurretik).	***Emaitza***: karaktere-taula (tamaina ezin da jakin aldez aurretik).
Mugak	***Adierazpena == Zenbakia1 ' ' Eragiketa ' ' Zenbakia2*** ***Zenbakia1***: zenbaki erromatar bat. ***Eragiketa*** $\in \{$'+', '–', '*'$\}$ ***Zenbakia2***: zenbaki erromatar bat.	***Emaitza***: zenbaki erromatar bat.
Erlazioak	Baldin ***Eragiketa ==*** '-' bada, orduan ***Z1 > Z2*** non ***Z1 Zenbakia1*** *zenbaki erromatarraren balioa baita eta* ***Z2 Zenbakia2*** *zenbaki erromatarraren balioa.*	Baldin ***Eragiketa ==*** '-' bada, orduan ***Em = Z1 – Z2*** Baldin ***Eragiketa ==*** '+' bada, orduan ***Em = Z1 + Z2*** Baldin ***Eragiketa ==*** '*' bada, orduan ***Em = Z1 * Z2*** non ***Z1 Zenbakia1*** zenbaki erromatarraren balioa baita, ***Z2 Zenbakia2*** zenbaki erromatarraren balioa baita, eta ***Em Emaitza*** zenbaki erromatarraren balioa.

Algoritmo nagusia

Bi zenbaki erromatarren eragiketa kalkulatzeko prozedurak aurki daitezke Interneten. Adibidez, 'VIII + IX' kalkulatzeko, lehendabizi kenketak dituzten zenbaki erromatarrak berridatzi behar ditugu kenketak batuketa bihurtuz. 'IX' ordez 'VIIII' izango dugu. Ondoren, bi zenbakien karaktereak ordenatuta elkarren ondoan jarri: 'VVIIIIIII'. Azkenik, multzoak ordezkatu: 'XVII'. Errepikatu behar adina aldiz, eta kenketak behar izanez gero jarri.

Kenketak badu beste prozedura bat, eta batuketak beste bat. Prozedura horiek algoritmo batean adieraz daitezke, baina konplexua da. Gure lana orain programa diseinatzea da, eta horretarako prozedura sinpleago bat topatzen badugu abantaila ugari izango ditugu. Alde batetik, arazoan eta konponbidean pentsatzea arinago izango zaigu, buruan errazago ikusten baititugu prozedura sinpleagoak. Bestetik, algoritmo sinpleago bat egiterakoan errore gutxiago sortuko dira, eta, sortu badira, aiseago aurkitu eta konponduko dira. Beraz, pentsa dezagun prozedura sinpleago batean.

Zenbaki arabiarren kalkulagailu bat erraz egin daiteke. Kalkulagailu erromatarra kalkulagailu arabiar bihurtuko bagenu, sinpleago izango litzateke. Horretarako zenbaki erromatarrak zenbaki arabiar bihur ditzakegu, eta alderantziz. Hala, problema konplexua zena problema sinpleagoen elkarlanarekin ebatz daiteke: erabiltzaileak emandako adierazpen matematikotik zenbaki erromatarrak eta eragiketa hartu; bi zenbaki erromatarrak arabiar bihurtu; bi zenbaki arabiar horiekin kalkulatu nahi den eragiketa; emaitza arabiarra zenbaki erromatar bihurtu. Algoritmo posible bat hemen agertzen da.



Programazio modularra aplikatuz, konplexua zen problema bat sinplifikatu egin da. Baina programa hori egikaritzeko, hurrengo problemen azpiprogramak egin behar dira:

1. ZUZENA: erabiltzaileak emandako adierazpen matematikoak zehaztapenak ezartzen dituen aurrebaldintzak betetzen dituen ala ez adierazi.
2. ZATITU: adierazpen matematikoa hiru zatitan banandu (zenbakia1, eragiketa eta zenbakia2) eta horietako bat bueltatu.
3. BIHURTUARABIAR: zenbaki erromatar baten balioa kalkulatu.
4. BIHURTUERROMATAR: zenbaki arabiar bat zenbaki erromatar bihurtu.

Problema hauek ebazteko pauso guztiak jarraitu behar dira (zehaztapena, algoritmoa, simulazio-taula, programa, eta probak), eta hurrengo ataletan egiten da hori. Baina demagun egingo direla eta beren lana ondo egingo dutela. Kalkulagailu nagusiaren simulazio-taula egin dezakegu kasu berezi bat (adierazpen okerra; adibidez, 'V – IX' ez du onartu behar) eta orokor bat (adierazpen zuzena; 'V + IX' adierazpenak 'XIV' emaitza eman beharko luke) probatzeko.

	#	Adierazpena	Zenbakia1	Z1	Zenbakia2	Z2	Eragiketa	Em	Emaitza	Pantaila
	1	?	?	?	?	?	?	?	?	?
	2	?	?	?	?	?	?	?	?	'Adierazi adie...'
	3	'V - IX'	?	?	?	?	?	?	?	
ez	4	'V - IX'	?	?	?	?	?	?	?	
	2	'V - IX'	?	?	?	?	?	?	?	'Adierazi adie...'
	3	'V + IX'	?	?	?	?	?	?	?	
bai	4	'V + IX'	?	?	?	?	?	?	?	
	5	'V + IX'	'V'	?	?	?	?	?	?	
	6	'V + IX'	'V'	?	'IX'	?	?	?	?	
	7	'V + IX'	'V'	?	'IX'	?	'+'	?	?	
	8	'V + IX'	'V'	5	'IX'	?	'+'	?	?	
	9	'V + IX'	'V'	5	'IX'	9	'+'	?	?	
bai	10	'V + IX'	'V'	5	'IX'	9	'+'	?	?	
	11	'V + IX'	'V'	5	'IX'	9	'+'	14	?	
	15	'V + IX'	'V'	5	'IX'	9	'+'	14	'XIV'	
	16	'V + IX'	'V'	5	'IX'	9	'+'	14	'XIV'	'XIV'

Bi kasuak aldi berean probatu dira, eta ondo dabil algoritmoa. Kenketaren emaitza negatiboa ekidin da, eta baturaren emaitza zuzen kalkulatu da. Azpialgoritmoak egin gabe dauden arren, deitzen zaienean emaitza zuzena bueltatzen dutela suposa dezakegu. Hurrengo ataletan, azpialgoritmoek ebatzi beharreko problema txikiak ebazten dira.

Adierazpen zuzena azpiprograma

Programa honek karaktere-taula bat adierazpen erromatar matematiko zuzena den ala ez adierazi beharko du. Zehaztapena hau izan daiteke:

	Aurrebaldintza	Ondorengo baldintza
Zer	**Izangaia**: analizatu beharreko karaktere-taula.	***Epaia:*** sarrerako karaktere katea adierazpen matematiko erromatarra den ala ez.
Mota	**Izangaia**: karaktere-taula (tamaina ezin da jakin aldez aurretik).	***Epaia***: balio logikoa, bai ala ez.
Mugak		
Erlazioak		Baldin (**Izangaia** *==* ***Zenbakia1*** *‘ ‘* ***Eragiketa*** *‘ ‘* ***Zenbakia2*** ***Zenbakia1***: zenbaki erromatar bat ***Eragiketa*** $\in$ *{‘+’, ‘-’, ‘*’}* ***Zenbakia2***: zenbaki erromatar bat) eta baldin (***Eragiketa*** *==* ‘-’ bada, orduan ***Z1*** *>* ***Z2*** non ***Z1 Zenbakia1*** *zenbaki erromatarraren balioa baita eta* ***Z2 Zenbakia2*** *zenbaki erromatarraren balioa*) orduan ***Epaia*** *== true,* bestela ***Epaia*** *== false*

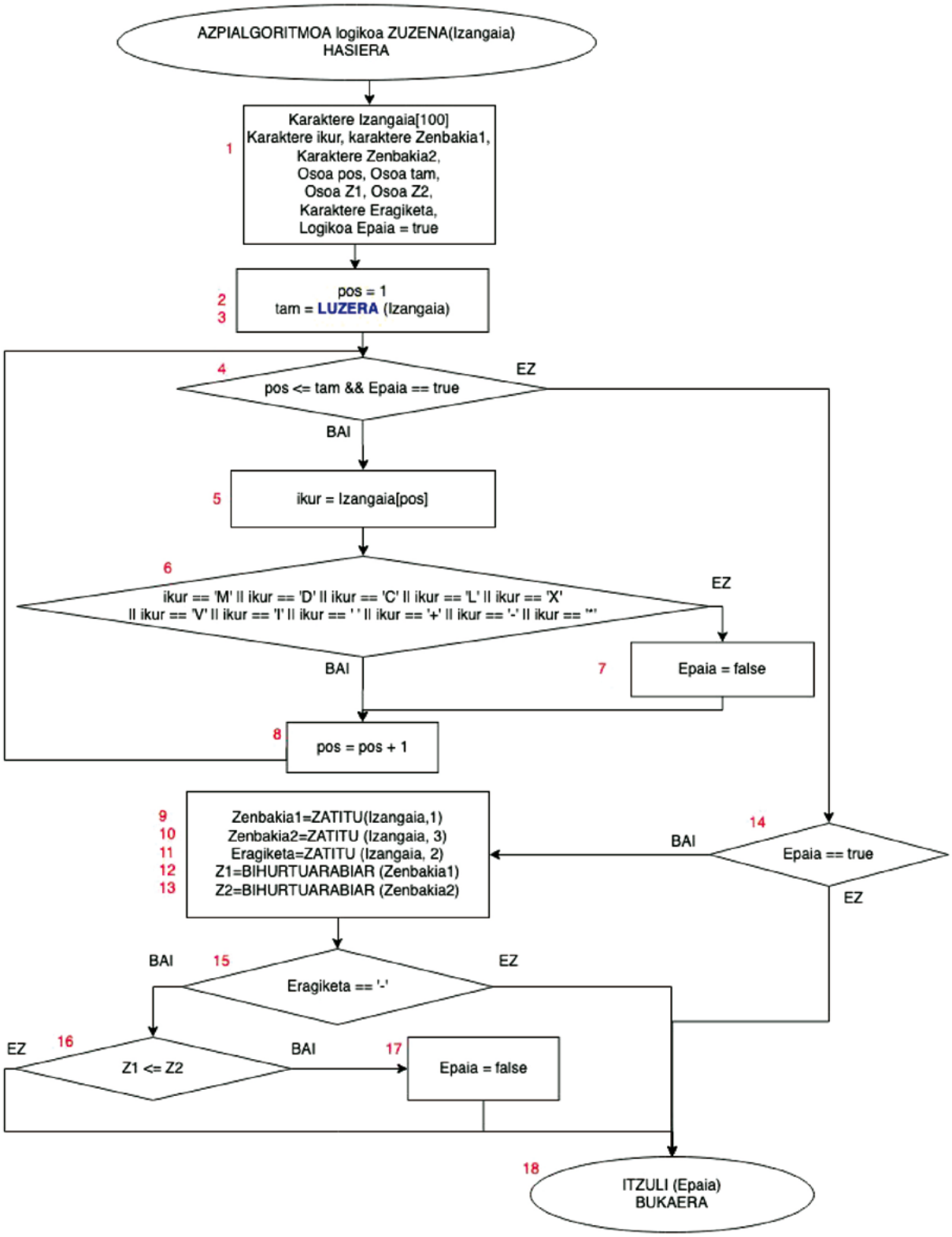
Adierazpenak zenbaki erromatar zuzenak dituen baieztatzea konplexua izan daiteke. Sinplifikatuz, azpialgoritmo honetan baieztatzen dena da zenbaki erromatarren karaktereak (‘I’, ‘V’, ‘X’, ‘L’, ‘C’, ‘D’ eta ‘M’), hutsuneak eta eragiketen karaktereak erabiltzen direla. Hori betez gero, eragiketa kenketa bada, lehen zenbakia bigarrena baino handiagoa izatea baieztatuko da.

Nabarmentzekoa da hurrengo orrialdean dagoen azpialgoritmoak beste azpialgoritmoei deitzen diela. ZATITU eta BIHURTUARABIAR azpialgoritmoak, aurrerago landuko direnez, erabil daitezke, algoritmo nagusian egin den moduan. Izan ere, azpialgoritmo batek beste azpialgoritmoei deitu diezaieke.

Programazio modularraren abantailak nabarmentzen ditu ariketa honek. Azpiprogramak behin egiten dira, eta askotan berrerabiltzen dira, denbora aurreztuz eta problema konplexuak sinplifikatuz.

LUZERA azpialgoritmoak karaktere-taularen tamaina bueltatu beharko luke. Azpialgoritmo hau ez dugu egingo, zeren programazio-lengoaiek izaten dute taula baten tamaina kalkulatzen duen azpiprogramaren bat. Beraz, existitzen dela suposatuko dugu.

‘V – IX’ adierazpenarekin probatuko da ZUZENA azpialgoritmo hau. Kasu orokor eta berezi denak probatu beharko genituzke, baina, eskuliburua gehiegi ez luzatzeko, bakar bat ikusiko da.


AZPIALGORITMOA logikoa ZUZENA(Izangaia)
HASIERA
1
Karaktere Izangaia[100]
Karaktere ikur, karaktere Zenbakia1,
Karaktere Zenbakia2,
Osoa pos, Osoa tam,
Osoa Z1, Osoa Z2,
Karaktere Eragiketa,
Logikoa Epaia = true
2
3
pos = 1
tam = LUZERA (Izangaia)
4
pos <= tam && Epaia == true
EZ
BAI
5
ikur = Izangaia[pos]
6
ikur == 'M' || ikur == 'D' || ikur == 'C' || ikur == 'L' || ikur == 'X'
|| ikur == 'V' || ikur == 'I' || ikur == ' ' || ikur == '+' || ikur == '-' || ikur == '*'
EZ
BAI
7
Epaia = false
8
pos = pos + 1
9
10
11
12
13
Zenbakia1=ZATITU(Izangaia,1)
Zenbakia2=ZATITU (Izangaia, 3)
Eragiketa=ZATITU (Izangaia, 2)
Z1=BIHURTUARABIAR (Zenbakia1)
Z2=BIHURTUARABIAR (Zenbakia2)
14
BAI
Epaia == true
EZ
BAI
15
Eragiketa == '-'
EZ
EZ
16
Z1 <= Z2
BAI
17
Epaia = false
18
ITZULI (Epaia)
BUKAERA

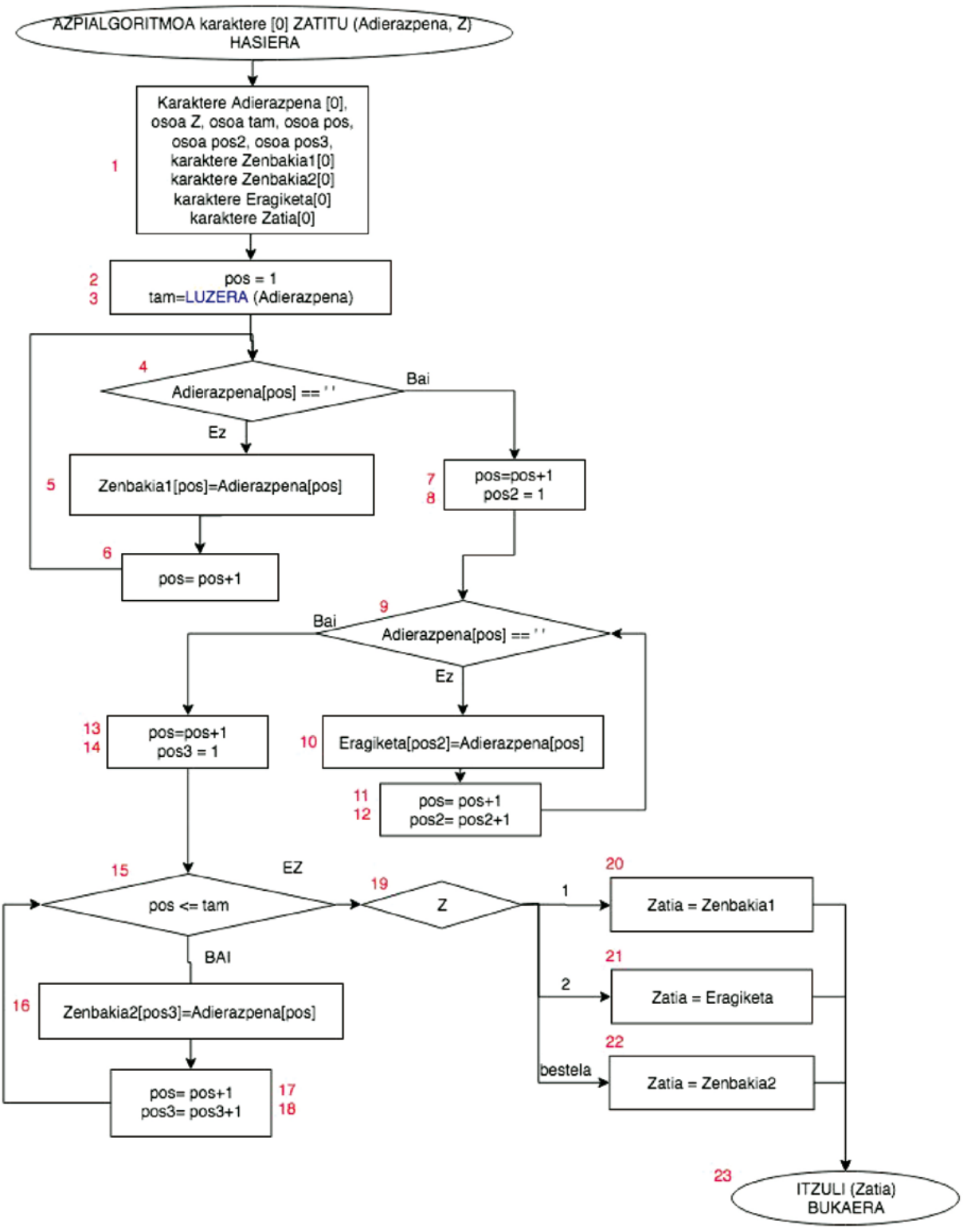
	#	Izangai	ikur	pos	tam	Zenbakia1	Z1	Zenbakia2	Z2	Eragiketa	Epaia
	1	'V - IX'	?	?	?	?	?	?	?	?	true
	2	'V - IX'	?	1	?	?	?	?	?	?	true
	3	'V - IX'	?	1	6	?	?	?	?	?	true
bai	4	'V - IX'	?	1	6	?	?	?	?	?	true
	5	'V - IX'	'V'	1	6	?	?	?	?	?	true
bai	6	'V - IX'	'V'	1	6	?	?	?	?	?	true
	8	'V - IX'	'V'	2	6	?	?	?	?	?	true
bai	4	'V - IX'	'V'	2	6	?	?	?	?	?	true
	5	'V - IX'	' '	2	6	?	?	?	?	?	true
bai	6	'V - IX'	' '	2	6	?	?	?	?	?	true
	8	'V - IX'	' '	3	6	?	?	?	?	?	true
bai	4	'V - IX'	' '	3	6	?	?	?	?	?	true
	5	'V - IX'	'-'	3	6	?	?	?	?	?	true
bai	6	'V - IX'	'-'	3	6	?	?	?	?	?	true
	8	'V - IX'	'-'	4	6	?	?	?	?	?	true
bai	4	'V - IX'	'-'	4	6	?	?	?	?	?	true
	5	'V - IX'	' '	4	6	?	?	?	?	?	true
bai	6	'V - IX'	' '	4	6	?	?	?	?	?	true
	8	'V - IX'	' '	5	6	?	?	?	?	?	true
bai	4	'V - IX'	' '	5	6	?	?	?	?	?	true
	5	'V - IX'	'I'	5	6	?	?	?	?	?	true
bai	6	'V - IX'	'I'	5	6	?	?	?	?	?	true
	8	'V - IX'	'I'	6	6	?	?	?	?	?	true
bai	4	'V - IX'	'I'	6	6	?	?	?	?	?	true
	5	'V - IX'	'X'	6	6	?	?	?	?	?	true
	6	'V - IX'	'X'	6	6	?	?	?	?	?	true
	8	'V - IX'	'X'	7	6	?	?	?	?	?	true
ez	4	'V - IX'	'X'	7	6	?	?	?	?	?	true
bai	14	'V - IX'	'X'	7	6	?	?	?	?	?	true
	9	'V - IX'	'X'	7	6	'V'	?	?	?	?	true
	10	'V - IX'	'X'	7	6	'V'	?	'IX'	?	?	true
	11	'V - IX'	'X'	7	6	'V'	?	'IX'	?	'-'	true
	12	'V - IX'	'X'	7	6	'V'	5	'IX'	?	'-'	true
	13	'V - IX'	'X'	7	6	'V'	5	'IX'	9	'-'	true
bai	15	'V - IX'	'X'	7	6	'V'	5	'IX'	9	'-'	true
bai	16	'V - IX'	'X'	7	6	'V'	5	'IX'	9	'-'	true
	17	'V - IX'	'X'	7	6	'V'	5	'IX'	9	'-'	false
	18	'V - IX'	'X'	7	6	'V'	5	'IX'	9	'-'	false

Adierazpena zatitu azpiprograma

Adierazpen matematikoa hiru zatitan banandu daiteke: lehen zenbaki erromatarra, eragiketa, eta bigarren zenbaki erromatarra. Azpialgoritmo honek hori egiten du, tartean hutsuneak dituztela aprobetxatuz. Eskatutako hiru zati horietako bat bueltatuko du.

	Aurrebaldintza	**Ondorengo baldintza**
Zer	***Adierazpena***: adierazpen matematikoa zenbaki erromatarrekin. ***Z***: zein zati jaso nahi duen.	***Zatia:*** adierazpenaren hiru zatietako bat.
Mota	***Adierazpena***: karaktere-taula (tamaina ezin da aldez aurretik jakin). ***Z***: zenbaki osoa.	***Zatia***: karaktere-taula (tamaina ezin da aldez aurretik jakin).
Mugak	***Adierazpena*** == ***Zenbakia1*** ‘ ‘ ***Eragiketa*** ‘ ‘ ***Zenbakia2*** ***Zenbakia1***: zenbaki erromatar zuzena. ***Eragiketa*** ∈ {‘+’, ‘–’, ‘*’} ***Zenbakia2***: zenbaki erromatar zuzena. ***Z*** ∈ {1,2,3}	
Erlazioak		Baldin **Z**==1 bada, orduan ***Zatia*** == ***Zenbakia1*** Baldin **Z**==2 bada, orduan ***Zatia*** == ***Emaitza*** Baldin **Z**==3 bada, orduan ***Zatia*** == ***Zenbakia2***

Algoritmoa hau izan daiteke.

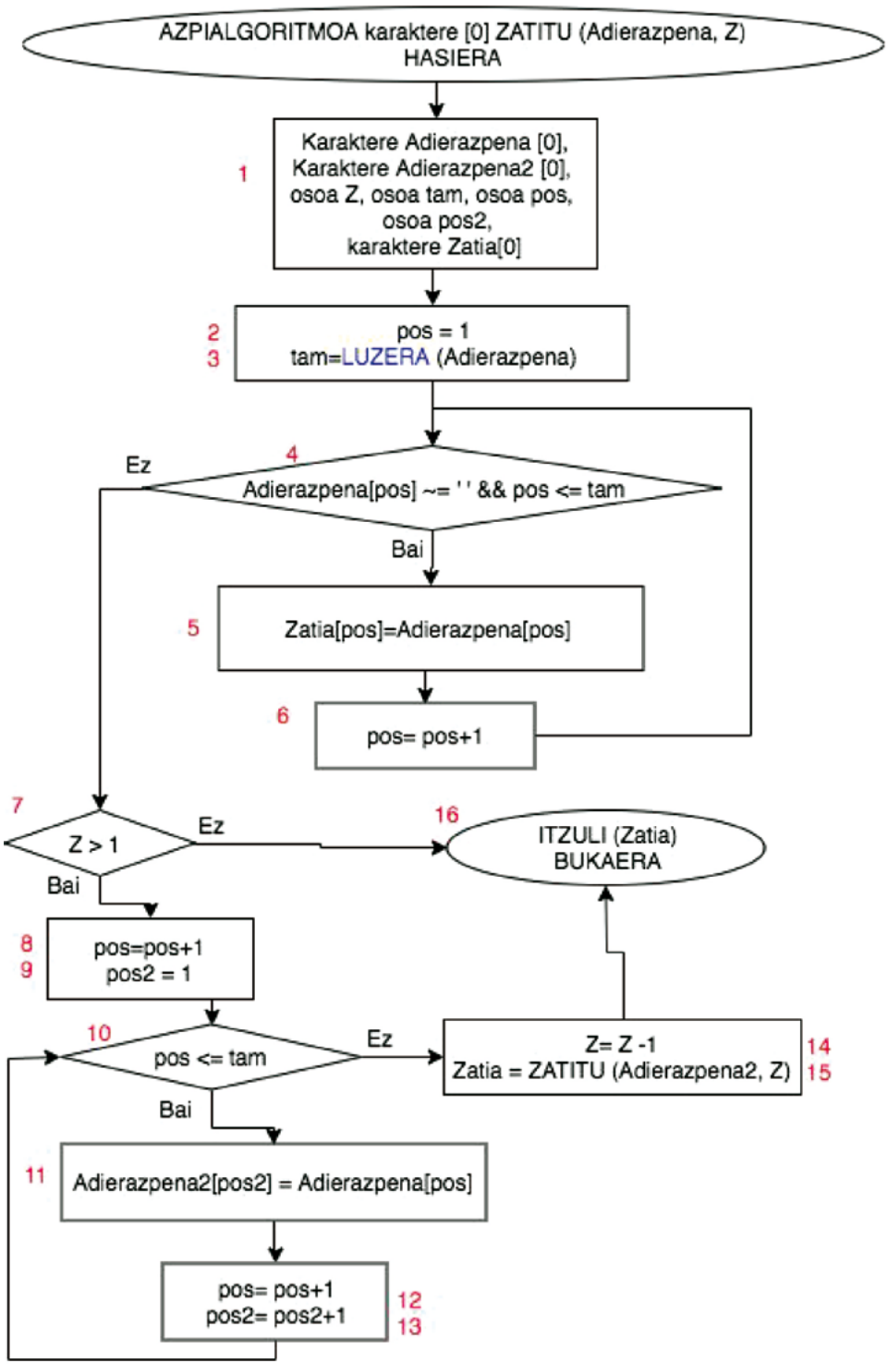


Kasu bakar bat probatuko da, adierazpena ‘V – IX’ izanda 3. zatia eskatzea. ‘IX’ bueltatu beharko luke. Simulazio-taula erakusten da jarraian.

	#	Adierazpena	Z	tam	pos	pos2	pos3	Zenbakia1	Zenbakia2	Eragiketa	Zatia
	1	'V - IX'	3	?	?	?	?	?	?	?	?
	2	'V - IX'	3	?	1	?	?	?	?	?	?
	3	'V - IX'	3	6	1	?	?	?	?	?	?
ez	4	'V - IX'	3	6	1	?	?	?	?	?	?
	5	'V - IX'	3	6	1	?	?	'V'	?	?	?
	6	'V - IX'	3	6	2	?	?	'V'	?	?	?
bai	4	'V - IX'	3	6	2	?	?	'V'	?	?	?
	7	'V - IX'	3	6	3	?	?	'V'	?	?	?
	8	'V - IX'	3	6	3	1	?	'V'	?	?	?
ez	9	'V - IX'	3	6	3	1	?	'V'	?	?	?
	10	'V - IX'	3	6	3	1	?	'V'	?	'-'	?
	11	'V - IX'	3	6	4	1	?	'V'	?	'-'	?
	12	'V - IX'	3	6	4	2	?	'V'	?	'-'	?
bai	9	'V - IX'	3	6	4	2	?	'V'	?	'-'	?
	13	'V - IX'	3	6	5	2	?	'V'	?	'-'	?
	14	'V - IX'	3	6	5	2	1	'V'	?	'-'	?
bai	15	'V - IX'	3	6	5	2	1	'V'	?	'-'	?
	16	'V - IX'	3	6	5	2	1	'V'	'I'	'-'	?
	17	'V - IX'	3	6	6	2	1	'V'	'I'	'-'	?
	18	'V - IX'	3	6	6	2	2	'V'	'I'	'-'	?
bai	15	'V - IX'	3	6	6	2	2	'V'	'I'	'-'	?
	16	'V - IX'	3	6	6	2	2	'V'	'IX'	'-'	?
	17	'V - IX'	3	6	7	2	2	'V'	'IX'	'-'	?
	18	'V - IX'	3	6	7	2	3	'V'	'IX'	'-'	?
ez	15	'V - IX'	3	6	7	2	3	'V'	'IX'	'-'	?
3	19	'V - IX'	3	6	7	2	3	'V'	'IX'	'-'	?
	22	'V - IX'	3	6	7	2	3	'V'	'IX'	'-'	'IX'
	23	'V - IX'	3	6	7	2	3	'V'	'IX'	'-'	'IX'

Esan bezala, azpialgoritmo batek beste bati dei diezaioke. Baita bere buruari ere, dei errekurtsiboa sortuz. ZATITU azpialgoritmoa errekurtsioaren bitartez ebatziko dugu, bi alternatibak konparatu ahal izateko: aurreko azpialgoritmo iteratiboa, eta hurrengo azpialgoritmo errekurtsiboa.

Zehaztapenak ezartzen duena betetzeko, azpialgoritmo errekurtsiboa hau egin dezakegu:



Azpialgoritmo honek jasotako adierazpenari lehen zatia kentzen dio. Hori eskatu bada (*Z* bat denean), bueltatu egiten du. Bestela, adierazpenean geratzen den beste guztia berriro ZATITU azpialgoritmoari ematen dio, Z-1 zatia nahi dela esanez. Horrela, Z bat balioa izan arte.

Ikus dezagun simulazio-taula, adierazpena ‘V – IX’ izanda eta 3. zatia eskatuz. ‘IX’ bueltatu beharko luke. Simulazio-taula hemen erakusten da.

ZATITU ('V - IX', 3)									
	#	Adierazpena	Z	Adierazpena2	tam	pos	pos2	Zatia	
	1	'V - IX'	3	?	?	?	?	?	
	2	'V - IX'	3	?	?	1	?	?	
	3	'V - IX'	3	?	6	1	?	?	
bai	4	'V - IX'	3	?	6	1	?	?	
	5	'V - IX'	3	?	6	1	?	'V'	
	6	'V - IX'	3	?	6	2	?	'V'	
ez	4	'V - IX'	3	?	6	2	?	'V'	
bai	7	'V - IX'	3	?	6	2	?	'V'	
	8	'V - IX'	3	?	6	3	?	'V'	
	9	'V - IX'	3	?	6	3	1	'V'	
bai	10	'V - IX'	3	?	6	3	1	'V'	
	11	'V - IX'	3	'-'	6	3	1	'V'	
	12	'V - IX'	3	'-'	6	4	1	'V'	
	13	'V - IX'	3	'-'	6	4	2	'V'	
bai	10	'V - IX'	3	'-'	6	4	2	'V'	
	11	'V - IX'	3	'- '	6	4	2	'V'	
	12	'V - IX'	3	'- '	6	5	2	'V'	
	13	'V - IX'	3	'- '	6	5	3	'V'	
bai	10	'V - IX'	3	'- '	6	5	3	'V'	
	11	'V - IX'	3	'- I'	6	5	3	'V'	
	12	'V - IX'	3	'- I'	6	6	3	'V'	
	13	'V - IX'	3	'- I'	6	6	4	'V'	
bai	10	'V - IX'	3	'- I'	6	6	4	'V'	
	11	'V - IX'	3	'- IX'	6	6	4	'V'	
	12	'V - IX'	3	'- IX'	6	7	4	'V'	
	13	'V - IX'	3	'- IX'	6	7	5	'V'	
ez	10	'V - IX'	3	'- IX'	6	7	5	'V'	
	14	'V - IX'	2	'- IX'	6	7	5	'V'	
	15	'V - IX'	2	'- IX'	6	7	5	'IX'	ZATITU('- IX', 2)
	16	'V - IX'	2	'- IX'	6	7	5	'IX'	

Hamabost aginduak ZATITU azpialgoritmoari deitzen dio ‘ – IX’ eta 2 datuekin. Horrek ZATITUren beste exekuzioa jartzen du martxan, eta ‘IX’ itzultzen dio. Hori lortzeko, beste dei errekurtsibo bat ere egiten da: ZATITU (‘IX’, 1). Hemen ikus daiteke horien simulazio-taula. Nabarmentzekoa da exekuzio errekurtsibo bakoitzak ordenagailuko memorian aldagai propioak dituela balio desberdinekin.

ZATITU ('- IX', 2)

	#	Adierazpena	Z	Adierazpena2	tam	pos	pos2	Zatia	
	1	'- IX'	2	?	?	?	?	?	
	2	'- IX'	2	?	?	1	?	?	
	3	'- IX'	2	?	4	1	?	?	
bai	4	'- IX'	2	?	4	1	?	?	
	5	'- IX'	2	?	4	1	?	'-'	
	6	'- IX'	2	?	4	2	?	'-'	
ez	4	'- IX'	2	?	4	2	?	'-'	
bai	7	'- IX'	2	?	4	2	?	'-'	
	8	'- IX'	2	?	4	3	?	'-'	
	9	'- IX'	2	?	4	3	1	'-'	
bai	10	'- IX'	2	?	4	3	1	'-'	
	11	'- IX'	2	'I'	4	3	1	'-'	
	12	'- IX'	2	'I'	4	4	1	'-'	
	13	'- IX'	2	'I'	4	4	2	'-'	
bai	10	'- IX'	2	'I'	4	4	2	'-'	
	11	'- IX'	2	'IX'	4	4	2	'-'	
	12	'- IX'	2	'IX'	4	5	2	'-'	
	13	'- IX'	2	'IX'	4	5	3	'-'	
ez	10	'- IX'	2	'IX'	4	5	3	'-'	
	14	'- IX'	1	'IX'	4	5	3	'-'	
	15	'- IX'	1	'IX'	4	5	3	'IX'	ZATITU('IX', 1)
	16	'- IX'	1	'IX'	4	5	3	'IX'	

ZATITU ('IX', 1)

	#	Adierazpena	Z	Adierazpena2	tam	pos	pos2	Zatia
	1	'IX'	1	?	?	?	?	?
	2	'IX'	1	?	?	1	?	?
	3	'IX'	1	?	2	1	?	?
bai	4	'IX'	1	?	2	1	?	?
	5	'IX'	1	?	2	1	?	'I'
	6	'IX'	1	?	2	2	?	'I'
bai	4	'IX'	1	?	2	2	?	'I'
	5	'IX'	1	?	2	2	?	'IX'
	6	'IX'	1	?	2	3	?	'IX'
ez	4	'IX'	1	?	2	3	?	'IX'
ez	7	'IX'	1	?	2	3	?	'IX'
	16	'IX'	1	?	2	3	?	'IX'

Zein da hobea? Azpialgoritmo iteratiboa ala errekurtsiboa? Biak dira balizkoak. Programatzaile-lanetan dabilena, erabaki beharko du zein ulertzen duen hoberen.

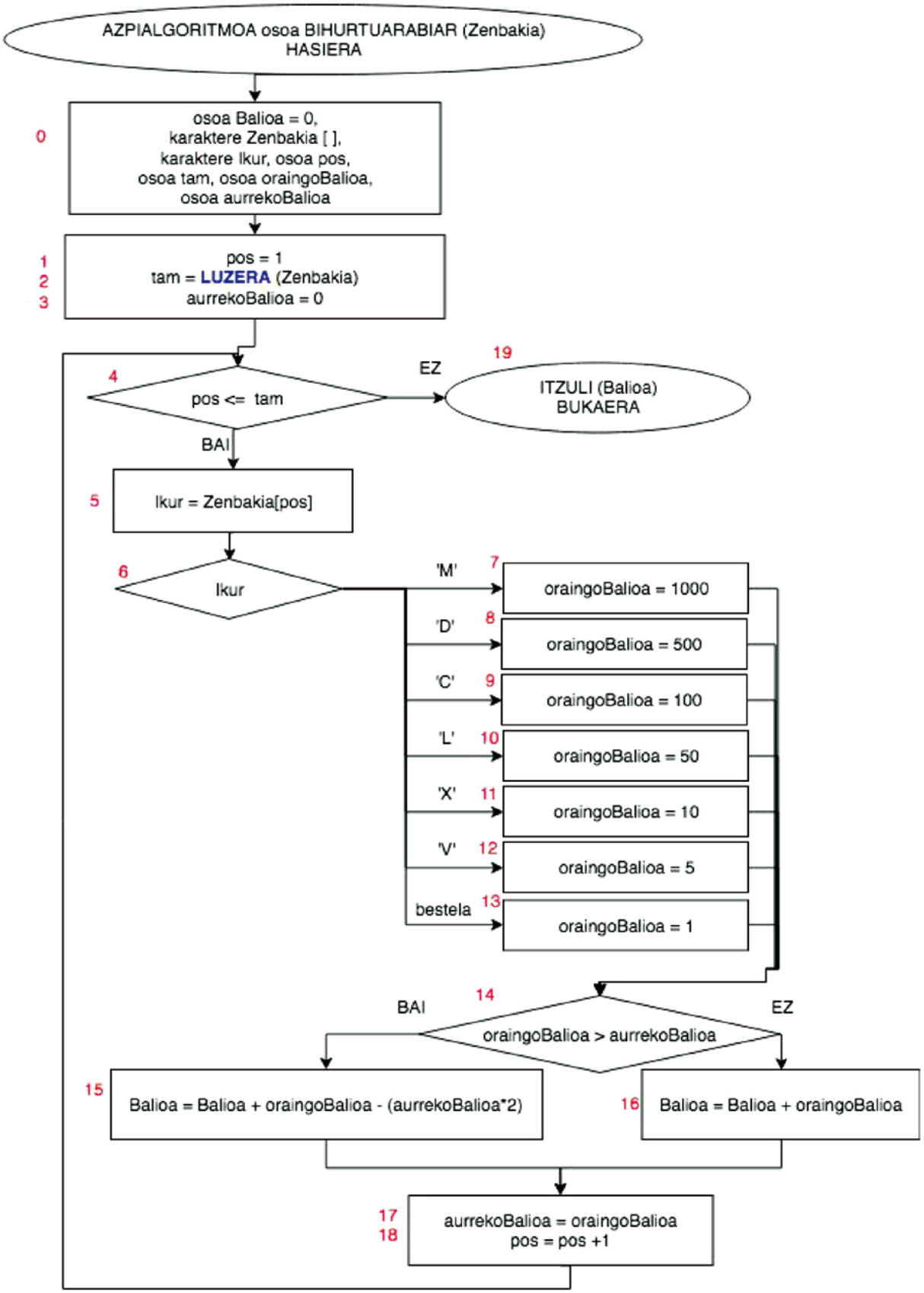
Hala ere, kasu honetan, azpialgoritmo errekurtsiboak abantaila bat izan dezake. Demagun mota honetako adierazpen matematiko konplexuagoak onartzen ditugula: 'III + IV + XXII + MDC'. Azpialgoritmo iteratiboa aldatu beharko genuke horrelako adierazpenak lantzeko. Aldiz, azpialgoritmo errekurtsiboa ez genuke aldatu behar, eta emaitza zuzena bueltatuko luke. Izan ere, errekurtsibitateak errepikatzen diren patroietatik ateratzen du bere abantaila. Adibide honetan, patroia da: <zatiBat> <hutsunea> <besteZatiak>. Patroi hau <besteZatiak> elementuan errepikatzen da, zati bakarra izan arte.

Bihurtu arabiar azpiprograma

Azpialgoritmo honek zenbaki erromatarraren balioa kalkulatuko du.

	Aurrebaldintza	Ondorengo baldintza
Zer	***Zenbakia***: zenbaki erromatar bat.	***Balioa***: zenbaki erromatarra arabiar bihurtuta.
Mota	***Zenbakia***: karaktere-taula (aldez aurretik ezin da jakin bere tamaina).	***Balioa***: zenbaki osoa.
Mugak	***Zenbakia***: zenbaki erromatarra.	***Balioa*** > 0
Erlazioak		***Balioa*** == Zenbakia-ren balioa, non: 'I' = 1, 'IV' = 4, 'V' = 5, 'IX' = 9, 'X' = 10, 'CL' = 40, 'L' = 50, 'XC' = 90, 'C' = 100, 'CD' = 400, 'D' = 500, 'XM' = 900, 'M' = 1.000 balio baitute.

Azpialgoritmoan zenbaki erromatarraren karaktereak korritzen dira, dagokion balioa gehitzeko. Baina aurreko karakterearen balioa momentukoa baino txikiagoa bada, aurreko balioaren bikoitza kentzen da. Adibidez, 'IX' zenbaki erromatarrean lehendabizi 'I' karakterea ikusten da, 1 balioa gehituz. Ondoren, 'X' karaktereak 10 batuko du. Baina, horrela 11 izango genuke. Aurreko balioa (1) oraingoarena baino txikiagoa denez (10), bere aurrekoaren bikoitza kentzen dugu (–2). Beraz, 9 balioa dugu (1 + 10 – 2). Era horretan, kalkulu zuzena egingo da.


AZPIALGORITMOA osoa BIHURTUARABIAR (Zenbakia)
HASIERA
0
osoa Balioa = 0,
karaktere Zenbakia [ ],
karaktere Ikur, osoa pos,
osoa tam, osoa oraingoBalioa,
osoa aurrekoBalioa
1
2
3
pos = 1
tam = LUZERA (Zenbakia)
aurrekoBalioa = 0
4
pos <= tam
EZ
19
ITZULI (Balioa)
BUKAERA
BAI
5
Ikur = Zenbakia[pos]
6
Ikur
'M'
7
oraingoBalioa = 1000
'D'
8
oraingoBalioa = 500
'C'
9
oraingoBalioa = 100
'L'
10
oraingoBalioa = 50
'X'
11
oraingoBalioa = 10
'V'
12
oraingoBalioa = 5
bestela
13
oraingoBalioa = 1
14
BAI
oraingoBalioa > aurrekoBalioa
EZ
15
Balioa = Balioa + oraingoBalioa - (aurrekoBalioa*2)
16
Balioa = Balioa + oraingoBalioa
17
18
aurrekoBalioa = oraingoBalioa
pos = pos +1

‘IX’ zenbaki erromatarrarekin probatuko da azpialgoritmoa.

	#	Zenbakia	Balioa	tam	pos	Ikur	oraingoBalioa	aurrekoBalioa
	0	'IX'	0	?	?	?	?	?
	1	'IX'	0	?	1	?	?	?
	2	'IX'	0	2	1	?	?	?
	3	'IX'	0	2	1	?	?	0
bai	4	'IX'	0	2	1	?	?	0
	5	'IX'	0	2	1	'I'	?	0
'I'	6	'IX'	0	2	1	'I'	?	0
	13	'IX'	0	2	1	'I'	1	0
bai	14	'IX'	0	2	1	'I'	1	0
	15	'IX'	1	2	1	'I'	1	0
	17	'IX'	1	2	1	'I'	1	1
	18	'IX'	1	2	2	'I'	1	1
bai	4	'IX'	1	2	2	'I'	1	1
	5	'IX'	1	2	2	'X'	1	1
'X'	6	'IX'	1	2	2	'X'	1	1
	11	'IX'	1	2	2	'X'	10	1
bai	14	'IX'	1	2	2	'X'	10	1
	15	'IX'	9	2	2	'X'	10	1
	17	'IX'	9	2	2	'X'	10	10
	18	'IX'	9	2	3	'X'	10	10
bai	4	'IX'	9	2	3	'X'	10	10
	19	'IX'	9	2	3	'X'	10	10

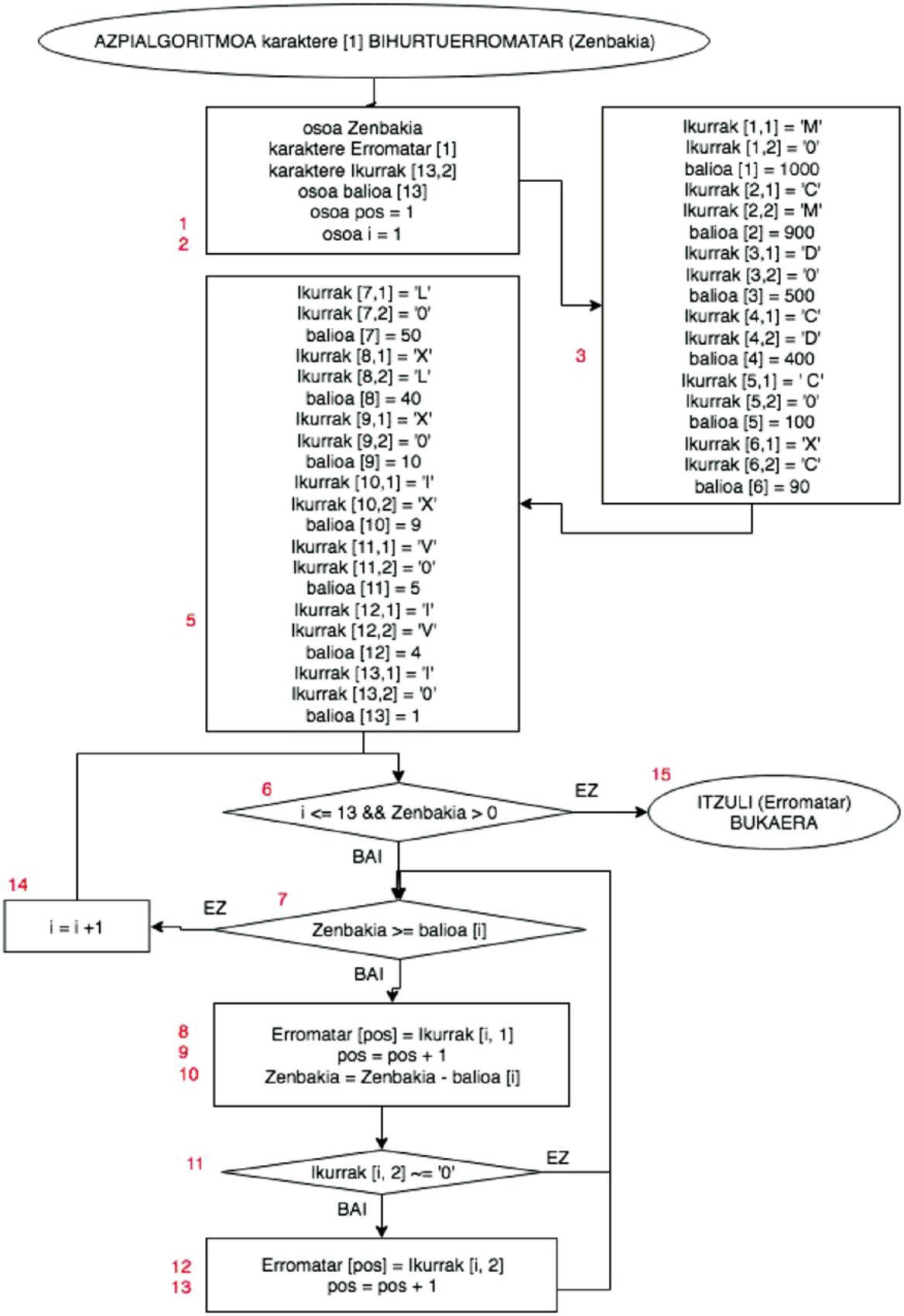
Bihurtu erromatar azpiprograma

Azpialgoritmo honek zenbaki oso bat erromatar bihurtzen du.

	Aurrebaldintza	Ondorengo baldintza
Zer	***Zenbakia***: zenbaki arabiar bat.	***Erromatar:*** sarrerako balioaren erromatar adierazpena.
Mota	***Zenbakia***: osoa.	***Erromatar***: karaktere-taula (aldez aurretik ezin da luzera jakin).
Mugak	***Zenbakia*** > 0	***Erromatar***: zenbaki erromatarra.
Erlazioak		***Erromatar*** zenbakiak adieraziko du ***Zenbakiaren*** balioa erromatarrez. *non:* 'I' = 1, 'IV' = 4, 'V' = 5, 'IX' = 9, 'X' = 10, 'CL' = 40, 'L' = 50, 'XC' = 90, 'C' = 100, 'CD' = 400, 'D' = 500, 'XM' = 900, 'M' = 1.000 balio baitute.

Azpialgoritmoak era honetan bihurtuko du zenbaki arabiarra erromatar. 1000 balioa baino handiagoa bada zenbakia, 'M' karakterea gehituko du emaitzara, eta 1000 kenduko dio zenbakiari. Hori behin eta berriro errepikatuko da, zenbakia 1000 baino txikiagoa den arte. Ondoren, 900 balioarekin konparatuko da; handiagoa bada 'XM' gehituko da emaitzara, eta 900 kendu zenbakiari. Prozedura berdina jarraituko da balio posible guztiekin, handienetik txikienera. Zenbakia azkenean zero izango da, eta emaitzan zenbaki erromatar baliokidea izango dugu.

Hemen agertzen da azpialgoritmoa, eta jarraian simulazio-taula. 1990 balioarekin probatuko da azpialgoritmoa. Emaitza zuzena 'MCMXC' da. Azpialgoritmo luzea denez, simulazio-taula ere horrelakoa izango zenez, sinplifikatu egin da. Simulazio-taularen eskuinaldean azpialgoritmoko 3 eta 5 aginduetan sortzen diren *Ikurrak* eta *balioa* taulak agertzen dira.


AZPIALGORITMOA karaktere [1] BIHURTUERROMATAR (Zenbakia)
osoa Zenbakia
karaktere Erromatar [1]
karaktere Ikurrak [13,2]
osoa balioa [13]
osoa pos = 1
osoa i = 1
1
2
Ikurrak [1,1] = 'M'
Ikurrak [1,2] = '0'
balioa [1] = 1000
Ikurrak [2,1] = 'C'
Ikurrak [2,2] = 'M'
balioa [2] = 900
Ikurrak [3,1] = 'D'
Ikurrak [3,2] = '0'
balioa [3] = 500
Ikurrak [4,1] = 'C'
Ikurrak [4,2] = 'D'
balioa [4] = 400
Ikurrak [5,1] = ' C'
Ikurrak [5,2] = '0'
balioa [5] = 100
Ikurrak [6,1] = 'X'
Ikurrak [6,2] = 'C'
balioa [6] = 90
3
Ikurrak [7,1] = 'L'
Ikurrak [7,2] = '0'
balioa [7] = 50
Ikurrak [8,1] = 'X'
Ikurrak [8,2] = 'L'
balioa [8] = 40
Ikurrak [9,1] = 'X'
Ikurrak [9,2] = '0'
balioa [9] = 10
Ikurrak [10,1] = 'I'
Ikurrak [10,2] = 'X'
balioa [10] = 9
Ikurrak [11,1] = 'V'
Ikurrak [11,2] = '0'
balioa [11] = 5
Ikurrak [12,1] = 'I'
Ikurrak [12,2] = 'V'
balioa [12] = 4
Ikurrak [13,1] = 'I'
Ikurrak [13,2] = '0'
balioa [13] = 1
5
6
i <= 13 && Zenbakia > 0
EZ
15
ITZULI (Erromatar)
BUKAERA
BAI
14
i = i +1
EZ
7
Zenbakia >= balioa [i]
BAI
8
9
10
Erromatar [pos] = Ikurrak [i, 1]
pos = pos + 1
Zenbakia = Zenbakia - balioa [i]
11
Ikurrak [i, 2] ~= '0'
EZ
BAI
12
13
Erromatar [pos] = Ikurrak [i, 2]
pos = pos + 1

	#	Zenbakia	Erromatar	pos	i
	1	1990	?	1	?
	2	1990	?	1	1
	3	1990	?	1	1
	5	1990	?	1	1
bai	6	1990	?	1	1
bai	7	1990	?	1	1
	8	1990	'M'	1	1
	9	1990	'M'	2	1
	10	990	'M'	2	1
ez	11	990	'M'	2	1
ez	7	990	'M'	2	1
	14	990	'M'	2	2
bai	6	990	'M'	2	2
bai	7	990	'M'	2	2
	8	990	'MC'	2	2
	9	990	'MC'	3	2
	10	90	'MC'	3	2
bai	11	90	'MC'	3	2
	12	90	'MCM'	3	2
	13	90	'MCM'	4	2
ez	7	90	'MCM'	4	2
	14	90	'MCM'	4	3
bai	6	90	'MCM'	4	3
ez	7	90	'MCM'	4	3
	14	90	'MCM'	4	4
bai	6	90	'MCM'	4	4
ez	7	90	'MCM'	4	4
	14	90	'MCM'	4	5
bai	6	90	'MCM'	4	5
ez	7	90	'MCM'	4	5
	14	90	'MCM'	4	6
bai	6	90	'MCM'	4	6
bai	7	90	'MCM'	4	6
	8	90	'MCMX'	4	6
	9	90	'MCMX'	5	6
	10	0	'MCMX'	5	6
bai	11	0	'MCMX'	5	6
	12	0	'MCMXC'	5	6
	13	0	'MCMXC'	6	6
ez	7	0	'MCMXC'	6	6
	14	0	'MCMXC'	6	7
ez	6	0	'MCMXC'	6	7
	15	0	'MCMXC'	6	7

	Ikurrak			balioa
1	'M'	'0'	1	1000
2	'C'	'M'	2	900
3	'D'	'0'	3	500
4	'C'	'D'	4	400
5	'C'	'0'	5	100
6	'X'	'C'	6	90
7	'L'	'0'	7	50
8	'X'	'L'	8	40
9	'X'	'0'	9	10
10	'I'	'X'	10	9
11	'V'	'0'	11	5
12	'I'	'V'	12	4
13	'I'	'0'	13	1

Programak

Kalkulagailu erromatarraren programa nagusia, azpiprogramak eta beren exekuzioen emaitzak hemen agertzen dira.

ErromatarKalkulagailua.m | ZUZENA.m | ZATITU.m | BIHURTUARABIAR.m | BIHURTUERROMATAR.m

```
1  function ErromatarKalkulagailua
2  Adierazpena = input ('Adierazi adierazpen erromatarra... ');
3  while ~ (ZUZENA (Adierazpena))
4     Adierazpena = input ('Adierazi adierazpen erromatarra... ');
5  end
6  Zenbakia1 = ZATITU (Adierazpena, 1);
7  Zenbakia2 = ZATITU (Adierazpena, 3);
8  Eragiketa = ZATITU (Adierazpena, 2);
9  Z1 = BIHURTUARABIAR (Zenbakia1);
10 Z2 = BIHURTUARABIAR (Zenbakia2);
11 if (Eragiketa=='+')
12    Em = Z1 + Z2;
13 else
14     if (Eragiketa=='-')
15         Em = Z1 - Z2;
16     else
17         Em = Z1 * Z2;
18     end
19 end
20 Emaitza = BIHURTUERROMATAR (Em);
21 disp (Emaitza)
22 end
```

Command Window

```
>> ErromatarKalkulagailua
Adierazi adierazpen erromatarra... 'V - IX'
Adierazi adierazpen erromatarra... 'V + IX'
XIV
fx >>
```

Bi kasu probatu dira egikaritza honetan: ‘V – IX’ eta ‘V + IX’.

ErromatarKalkulagailua.m | ZUZENA.m | ZATITU.m | BIHURTUARABIAR.m | BIHURTUERROMATAR.m

```
1   function Epaia = ZUZENA (Izangai)
2 -  Epaia = true;
3 -  pos = 1;
4 -  tam = length (Izangai);
5
6 -  while (pos <= tam && Epaia == true)
7 -      ikur = Izangai (pos);
8 -      if ~ (ikur == 'M' || ikur == 'D' || ikur == 'C' || ikur == 'L' ...
9                || ikur == 'X' || ikur == 'V' || ikur == 'I' || ikur == ' ' ...
10                   || ikur == '+' || ikur == '-' || ikur == '*')
11 -         Epaia = false;
12 -     end
13 -     pos = pos +1;
14 -  end
15 -  if ( Epaia == true)
16 -      Zenbakia1 = ZATITU(Izangai,1);
17 -      Zenbakia2 = ZATITU(Izangai,3);
18 -      Eragiketa = ZATITU(Izangai,2);
19 -      Z1 = BIHURTUARABIAR(Zenbakia1);
20 -      Z2 = BIHURTUARABIAR(Zenbakia2);
21 -      if (Eragiketa == '-')
22 -          if ~(Z1 > Z2)
23 -              Epaia = false;
24 -          end
25 -      end
26 -  end
27 -  end
```

Command Window

```
>> ZUZENA ('V - IX')

ans =

  logical

   0

>> 0 false adierazten du.
```

0 erantzuna, `false`, adierazten du. ‘V – IX’ ez da zuzena, emaitza negatiboa izango litzatekeelako.

ErromatarKalkulagailua.m | ZUZENA.m | ZATITU.m | BIHURTUARABIAR.m | BIHURTUERROMATAR.m

```
1  function Zatia = ZATITU (Adierazpena, Z)
2  pos = 1;
3  tam = length(Adierazpena);
4  while (~(Adierazpena(pos) == ' '))
5      Zenbakia1(pos) = Adierazpena(pos);
6      pos = pos+1;
7  end
8  pos = pos+1;
9  pos2 = 1;
10 while (~(Adierazpena(pos) == ' '))
11     Eragiketa(pos2) = Adierazpena(pos);
12     pos = pos+1;
13     pos2 = pos2+1;
14 end
15 pos = pos+1;
16 pos3 = 1;
17 while (pos <= tam)
18     Zenbakia2(pos3) = Adierazpena(pos);
19     pos = pos+1;
20     pos3 = pos3+1;
21 end
22 switch (Z)
23     case 1
24         Zatia = Zenbakia1;
25     case 2
26         Zatia = Eragiketa;
27     otherwise
28         Zatia = Zenbakia2;
29 end
30 end
```

Command Window

```
>> ZATITU ('V - IX', 3)

ans =

    'IX'
```

‘V – IX’ adierazpenaren hirugarren zatia hartzen da.

ErromatarKalkulagailua.m | ZUZENA.m | ZATITU1.m | ZATITU.m | BIHURTUARABIAR.m | BIHURTUERRO

```
1   function Zatia = ZATITU (Adierazpena, Z)
2 -  pos = 1;
3 -  tam = length(Adierazpena);
4 -  while (pos <= tam && Adierazpena(pos) ~= ' ')
5 -      Zatia(pos) = Adierazpena(pos);
6 -      pos = pos+1;
7 -  end
8 -  if (Z > 1)
9 -      pos = pos+1;
10 -     pos2 = 1;
11 -     while (pos <= tam)
12 -         Adierazpena2(pos2) = Adierazpena(pos);
13 -         pos = pos+1;
14 -         pos2 = pos2+1;
15 -     end
16 -     Z = Z - 1;
17 -     Zatia = ZATITU (Adierazpena2, Z);
18 - end
19 - end
```

Command Window

```
>> ZATITU ('V - IX', 3)

ans =

    'IX'

>> ZATITU ('III + IV + XXII + MDC', 5)

ans =

    'XXII'
```

Zatitu errekurtsiboa ere azaltzen da, nahi duenak erabil dezan. Bi exekuzio desberdin erakusten dira agindu-leihoan. ‘V – IX’ adierazpenaren hirugarren zatia hartzen da batean. Bestean, ‘III + IV + XXII + MDC’ adierazpenaren bosgarren zatia.

ErromatarKalkulagailua.m | ZUZENA.m | ZATITU1.m | ZATITU.m | BIHURTUARABIAR.m | BIHURTUERRO

```
1  function Balioa = BIHURTUARABIAR (Zenbakia)
2  Balioa = 0;
3  pos = 1;
4  tam= length (Zenbakia);
5  aurrekoBalioa = 0;
6  while (pos <=  tam)
7      Ikur = Zenbakia(pos);
8      switch (Ikur)
9          case 'M'
10             oraingoBalioa = 1000;
11         case 'D'
12             oraingoBalioa = 500;
13         case 'C'
14             oraingoBalioa = 100;
15         case 'L'
16             oraingoBalioa = 50;
17         case 'X'
18             oraingoBalioa = 10;
19         case 'V'
20             oraingoBalioa = 5;
21         otherwise
22             oraingoBalioa = 1;
23     end
24 if (oraingoBalioa > aurrekoBalioa)
25     Balioa = Balioa + oraingoBalioa - (aurrekoBalioa*2);
26 else
27     Balioa = Balioa + oraingoBalioa;
28 end
29 aurrekoBalioa = oraingoBalioa;
30 pos = pos +1;
31 end
32 end
```

Command Window

```
>> BIHURTUARABIAR ('IX')

ans =

     9
```

‘IX’ zenbaki erromatarra 9 zenbaki arabiar bihurtzen da.

ErromatarKalkulagailua.m | ZUZENA.m | ZATITU.m | BIHURTUARABIAR.m | BIHURTUERROMATAR.m

```
function Erromatar =  BIHURTUERROMATAR (Zenbakia)
pos=1;
i=1;
Ikurrak (1,1) = 'M';
Ikurrak (1,2) = '0';
balioa (1) = 1000;
Ikurrak (2,1) = 'C';
Ikurrak (2,2) = 'M';
balioa (2) = 900;
Ikurrak (3,1) = 'D';
Ikurrak (3,2) = '0';
balioa (3) = 500;
Ikurrak (4,1) = 'C';
Ikurrak (4,2) = 'D';
balioa (4) = 400;
Ikurrak (5,1) = 'C';
Ikurrak (5,2) = '0';
balioa (5) = 100;
Ikurrak (6,1) = 'X';
Ikurrak (6,2) = 'C';
balioa (6) = 90;
Ikurrak (7,1) = 'L';
Ikurrak (7,2) = '0';
balioa (7) = 50;
Ikurrak (8,1) = 'X';
Ikurrak (8,2) = 'L';
balioa (8) = 40;
Ikurrak (9,1) = 'X';
Ikurrak (9,2) = '0';
balioa (9) = 10;
Ikurrak (10,1) = 'I';
Ikurrak (10,2) = 'X';
balioa (10) = 9;
Ikurrak (11,1) = 'V';
Ikurrak (11,2) = '0';
balioa (11) = 5;
Ikurrak (12,1) = 'I';
Ikurrak (12,2) = 'V';
balioa (12) = 4;
Ikurrak (13,1) = 'I';
Ikurrak (13,2) = '0';
balioa (13) = 1;

while (i<=13 && Zenbakia > 0)
    while (Zenbakia >= balioa (i))
        Erromatar (pos) = Ikurrak (i, 1);
        pos = pos + 1;
        Zenbakia = Zenbakia - balioa (i);
        if (Ikurrak (i, 2) ~= '0')
            Erromatar (pos) = Ikurrak (i, 2);
            pos = pos + 1;
        end
    end
    i=i+1;
end
end
```

Command Window

```
>> BIHURTUERROMATAR (1990)

ans =

    'MCMXC'
```

Azkenik, 1990 zenbaki arabiarra zenbaki erromatar bihurtzen da.

Honekin amaitzen da eskuliburu hau. Informatikan sakondu nahi duenak bibliografia ataletan ageri diren erreferentziak azter ditzake.

Eskerrak

Liburu hau, Euskararen eta Etengabeko Prestakuntzaren arloko errektoreordetzaren (Euskal Herriko Unibertsitatea) euskarazko ikasmaterialgintza eta eskuliburugintza sustatzeko 2018ko deialdiaren barne, sortua izan da. Errektoreordetzak eskaini digun arreta eskertu nahi dugu. Bereziki, Larraitz Garmendia Munduateren berrikuspen lana. Bere orrazketa finak eta metodikoak, asko hobetu du liburu honen euskara maila.

Erreferentziak

Definizioen iturria

Euskalterm terminologia-banku publikoa: https://www.euskadi.eus/euskalterm/

Harluxet Hiztegi Entziklopedikoa: http://www1.euskadi.net/harluxet/

Wikipedia, Entziklopedia askea: https://eu.wikipedia.org/wiki/Azala

Programazio-lengoaiak

Alegría, Iñaki; Perez de Viñaspre Garralda, Olatz; Sarasola Gabiola, K. *Python Programazio-Lengoaia: Oinarriak Eta Aplikazioak*; Udako Euskal Unibertsitatea. Bilbo, 2016.

Ángel Franco García. *MATLAB para el Grado en Ingeniería de Energías Renovables (Webgunea)*; Gipuzkoako Ingenieritza Eskola. Eibarko atala (UPV/EHU). 2016. http://www.sc.ehu.es/sbweb/fisica3/intro.html

Bastarrika, J. R.; Sarasola, K. *Lisp: Programazio-Lengoaia*; Udako Euskal Unibertsitatea. Bilbo, 1991.

Castrillón Santana, Modesto. *Fundamentos De Informática Y Programación Para Ingeniería: Ejercicios Resueltos Para C Y Matlab*; Paraninfo. Madrid, 2011.

Goirizelaia Ordorika, Iñaki. *Programazioaren oinarriak*; Euskal Herriko Unibertsitateko Argitalpen Zerbitzua. 1999.

Mijangos, Eugenio. *Zenbakizko Metodoak Matlab Erabiliz*; Unibertsitateko Eskuliburuak. Euskal Herriko Unibertsitateko Argitalpen Zerbitzua. Leioa, 2017.

Euskara

EHULKUren aholkuak : https://www.ehu.eus/eu/web/euskara/ehulkuren-aholkuak

Elhuyar hiztegiak https://hiztegiak.elhuyar.eus

Euskal Jakintza baliabideak: http://euskaljakintza.com/baliabideak/

Glosbe - eleanitza online dictionary: https://eu.glosbe.com/

IVAP, Herri Arduralaritzaren Euskal Erakundea - Euskaraz lan egiteko baliabideak: http://www.ivap.euskadi.eus/euskaraz-lan-egiteko-baliabideak/

Sareko Euskal Gramatika: http://www.ehu.eus/seg/

UZEI, Terminologia eta Lexikografia Zentroa - Sinonimoen Hiztegia: http://www.uzei.eus/zerbitzuak-eta-produktuak/produktuen-katalogoa/sinonimoen-hiztegia/

Xuxen euskarazko zuzentzaile ortografiko eta gramatikalaren webgunea: http://xuxen.eus

5.000 adorez hiztegiak: http://www.bostakbat.org/azkue/

UNIBERTSITATEKO ESKULIBURUAK
MANUALES UNIVERSITARIOS

INFORMAZIOA ETA ESKARIAK • INFORMACIÓN Y PEDIDOS

UPV/EHUko Argitalpen Zerbitzua • Servicio Editorial de la UPV/EHU
argitaletxea@ehu.eus • editorial@ehu.eus
1397 Posta Kutxatila - 48080 Bilbo • Apartado 1397 - 48080 Bilbao
Tfn.: 94 601 2227 • www.ehu.eus/argitalpenak

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea