Kudeaketaren eta Informazio Sistemen Informatikaren
Ingeniaritzako Gradua

Bilboko Ingeniaritza Eskola
UPV / EHU

Lengoaiak eta Sistema Informatikoak Saila

PROGRAMAZIOAREN METODOLOGIA

1. maila

31 taldea

2018-19 ikasturtea

2. gaia
Programen espezifikazioa eta dokumentazioa

Irakaslea: José Gaintzarain Ibarmia

Bulegoa: P3I 40
Tutoretza-ordutegia: GAUR-en begiratu

2. gaia
Programen espezifikazioa eta dokumentazioa

2.1. Sarrera

2.1.1. Helburua

Programa bat garatzerakoan oso garrantzitsua da programak zer egin behar duen ondo zehaztea.

Programa batek zer egin behar duen adierazteko edo espezifikatzeko espezifikazio-lengoaia formal bat erabili behar da.

Espezifikazio-lengoaia formal bat erabiliz ez da anbiguotasunik egongo.

Gai honetako lehenengo helburua **Lehen Mailako Logika**ren lengoaia eta **aurre-ondoetako espezifikazioa** bezala ezagutzen den teknika erabiliz programak espezifikatzen ikastea da.

Bigarren helburua Lehen Mailako Logikaren lengoaia erabiliz programak dokumentatzea da, programen puntu desberdinetan betetzen diren propietateak azalduz.

2.1.2. Espezifikazioaren definizioa

"Espezifikazioa" kontzeptuaren definizioa agertzen da jarraian:

- Sistema baten ezaugarriak eta portaera era zehatzean adierazten dituen dokumentua.

2.1.3. Programen zuzentasuna

Programa edo algoritmo bat zuzena al den jakiteko, programaren sarrerako datuak kontuan hartuz irteerako emaitzek zein baldintza bete beharko lituzketen espezifikatu beharko da era zehatzean.

Programa bat zuzena izango da programak egiten duena eta egin beharko lukeena (espezifikazioa) bat datozenean.

2.2. Aurre-ondoetako espezifikazioa

2.2.1. Aurreko baldintza eta ondorengo baldintza

Programa batek zer egin beharko lukeen adierazteko aurre-ondoetako espezifikazioa erabiliko dugu.

Aurre-ondoetako espezifikazio batean programa exekutatzen hasi aurretik bete beharreko baldintza eta programa exekutatu ondoren bete beharreko baldintza edukiko ditugu.

Aurrebaldintzaren (edo **hasierako baldintza**ren) bidez sarrerako datuek bete beharko dituzten baldintzak zehazten dira. Programa exekutatzen hasi baino lehenago sarrerako datuek aurreko baldintza (edo hasierako baldintza) bete behar dute.

Ondorengo baldintzaren (edo **bukaerako baldintza**ren) bidez sarrerako datuen eta emaitzen arteko erlazioa zehaztuko da. Programaren exekuzioa bukatutakoan emaitzek ondorengo baldintza (edo bukaerako baldintza) bete beharko dute.

Aurre-ondoetako espezifikazio baten esanahia honako hau da: sarrerako datuek aurreko baldintza betetzen dutela suposatuz, programa exekutatutakoan emaitzek ondorengo baldintza bete beharko dute.

1. Adibidea:

Aurrebaldintza ≡ **{x ≥ 9}**
x : = x + 5;
Ondorengo baldintza ≡ **{x ≥ 14}**

Espezifikazio horren arabera, esleipenaren aurretik x-ren balioa 9 baino handiagoa edo berdina bada, esleipenaren ondoren x-ren balioa 14 baino handiagoa edo berdina izango da.

2. Adibidea:
Zenbaki osoz osatutako A(1..n) bektore ez-huts baten elementuen batura s aldagaian lortzen duen programari dagokion aurre-ondoetako espezifikazioa honako hau da:

Aurrebaldintza ≡ **{n ≥ 1}**
i : = 0; s : = 0;
while i < n loop
    i : = i + 1;
    s : = s + A(i);
end loop;
Ondorengo baldintza ≡ **{s =** $\sum_{k=1}^{n} A(k)$ **}**

2.2.2. Kontratu bidezko garapena

Aurre-ondoetako espezifikazioaren bidez programaren eta programa hori erabiliko duenaren artean kontratua finkatzen da:

- Aurrebaldintza (edo hasierako baldintza):
  - Programak ondo funtzionatzeko sarrerako datuek zein baldintza bete beharko dituzten adierazten du.
  - Programa exekutatzerakoan sarrerako datuek aurreko baldintzan zehaztutakoa betetzen dutela suposatuko da beti.
  - Programa ez da arduratuko aurrebaldintza betetzen ez duten datuez. Programaren barruan ez da egiaztatuko datuek aurreko baldintza betetzen al duten ala ez.
- Ondorengo baldintza (edo bukaerako baldintza):
  - Hasieran aurreko baldintza betetzen zela suposatuz, emaitzek zein baldintza beteko dituzten adierazten du programaren ondorengo baldintzak.

Programari aurrebaldintza betetzen duten datuak ematen bazaizkio, programak ondorengo baldintza edo bukaerako baldintza betetzen duten emaitzak itzuliko ditu.

2.2.3. Adibide gehiago

Aurrebaldintzari φ ('fi') deituko diogu normalean eta ondorengo baldintzari ψ ('psi') deituko diogu.

3. Adibidea:
Adibide honetan x eta y bi zenbaki emanda eta x-ren balioa y-rena baino handiagoa dela jakinda, y – x kenduraren balio absolutua z aldagaian kalkulatzen duen programaren aurre-ondoetako espezifikazioa daukagu:

{φ} ≡ {x ≥ y}
z : = x – y;
{ψ} ≡ {z = |y – x|}

4. Adibidea:
Adibide honetan x eta y bi zenbaki emanda, y – x kenduraren balio absolutua z aldagaian kalkulatzen duen programaren aurre-ondoetako espezifikazioa daukagu:

{φ} ≡ {True}
if x >= y **then** z : = x – y;
else z : = y – x; **end if**;
{ψ} ≡ {z = |y – x|}

Sarrerako datuek inolako baldintzarik ez dutela bete behar adierazteko, aurrebaldintza bezala 'True' formula ipintzen da.

5. Adibidea:
Jarraian x eta y aldagaien balioa trukatzen duen programari dagokion aurre-ondoetako espezifikazioa dator:

{φ} ≡ {x = *a* ∧ y = b}
x : = x + y;
y : = x – y;
x : = x – y;
{ψ} ≡ {x = b ∧ y = *a*}

Ez dakigu hasieran x eta y aldagaien balioa zein den, baina bukaeran balio horiek trukatuta daudela adierazi ahal izateko, izen bat eman behar zaie x eta y-ren hasierako balioei. Kasu honetan ***a*** eta **b** izenak erabili dira.

2.3. Lehen Mailako Logikaren Lengoaia

Aurreko ataleko adibideetan ikusi den bezala, programa baten aurre-ondoetako espezifikazioa idazteko Lehen Mailako Logikaren lengoaia erabiliko dugu.

Jarraian lengoaia hori osatzen duten elementuak aztertuko dira.

2.3.1. Alfabetoa

Formulak idazterakoan zein sinbolo erabil ditzakegun zehazten da alfabetoaren bidez:

- Konstanteak:
  - Zenbakiak: ..., -3, -2, -1, 0, 1, 2, 3, ...
  - Letrak: 'a', 'b', 'c', ..., 'z', ...
  - Beste konstante batzuk (edo konstanteen izenak): *a*, b, c, ...
  - Balio boolearrak: True, False
- Aldagaiak: x, y, z, s, zenb, batura, aldiz, ...
- Funtzioak (funtzio aritmetikoak): +, -, –, *, mod, div, ..., $\sum$, $\prod$, ...,
  (**Funtzio aritmetikoak** emaitza bezala zenbakizko balio bat itzultzen duten funtzioak dira)
- Predikatuak: =, $\neq$, <, $\leq$, >, $\geq$, bikoitia, bakoitia, ...
  (**Predikatuak** emaitza bezala balio boolear bat itzultzen duten funtzioak dira)
- Eragile logikoak: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- Zenbatzaile logikoak: $\exists$, $\forall$
- Parentesiak: (,)

2.3.2. Sintaxia

Sintaxiak formulak idazterakoan zein arau jarraitu behar diren adierazten digu. Arau horiek jarraituz eratutako formulak formula egokiak izango dira:

2.3.2.1. Terminoak

Honako elementu hauek **terminoak** direla esaten da:

- konstanteak: 4, 6, -6, 'a', 'b', ..., *a*, b, ...
- aldagaiak: x, y, z, r, batura, aldiz, ...
- terminoei aplikatutako funtzioak. Esate baterako, f funtzioa n argumentu behar dituen funtzio bat baldin bada eta t_1, t_2, ..., t_n terminoak baldin badira, orduan $f(t_1, t_2, ..., t_n)$ termino bat da.

  Adibidez, hand funtzioa bi zenbakiren arteko handiena itzultzen duen funtzioa baldin bada, orduan hand(5, 2) termino bat izango da, hand(y, z + 1) beste termino bat izango da eta abar.

2.3.2.2. Formulak

Jarraian aipatzen diren elementuak **formulak** dira:

- atomoak edo terminoei aplikatutako predikatuak. Esate baterako, p predikatua n argumentu behar dituen predikatu bat baldin bada eta t_1, t_2, ..., t_n terminoak baldin badira, orduan $p(t_1, t_2, ..., t_n)$ formula bat izango da.

  Adibidez, bikoitia izeneko predikatua zenbaki oso bat hartu eta zenbaki hori bikoitia bada True eta bestela False itzultzen duen predikatua da. Horrela, bikoitia(5) formula bat da, bikoitia(y + 2) beste formula bat da, eta abar.

- Eragile logikoak erabiliz formulak elkartuz lortzen diren espresioak ere formulak dira:
  - $\neg\varphi$ (**ez** fi)
  - $\varphi \wedge \psi$ (fi **eta** psi)
  - $\varphi \vee \psi$ (fi **edo** psi)
  - $\varphi \rightarrow \psi$ (**baldin** fi **orduan** psi)
  - $\varphi \leftrightarrow \psi$ (fi **baldin eta soilik baldin** psi)

  φ eta ψ formulak direla kontsideratuz

- Formulei zenbatzaile logikoak aplikatuz lortzen diren espresioak ere formulak dira:
  - $\exists x(\varphi)$ Zenbatzaile existentziala

    Esanahia: φ formulak dioena betetzen duen x balioren bat existitzen da (gutxienez bat existitzen da, baina ez dakigu zenbat diren).
  - $\forall x(\varphi)$ Zenbatzaile unibertsala

    Esanahia: x balio denek φ formulak dioena betetzen dute.

2.3.3. Semantika

Semantikak formula baten balioa zein den erabakitzeko arauak ematen dizkigu, hau da, formula bat egiazkoa al den ala ez (formula betetzen al den ala ez) nola erabaki adierazten digu.

2.3.3.1. $\neg$, $\wedge$, $\vee$, $\rightarrow$ eta $\leftrightarrow$ eragile logikoen semantika

φ	$\neg\varphi$
False	True
True	False

φ	ψ	$\varphi \wedge \psi$
False	False	False
False	True	False
True	False	False
True	True	True

φ	ψ	$\varphi \vee \psi$
False	False	False
False	True	True
True	False	True
True	True	True

φ	ψ	$\varphi \rightarrow \psi$
False	False	True
False	True	True
True	False	False
True	True	True

φ	ψ	$\varphi \leftrightarrow \psi$

False	False	True
False	True	False
True	False	False
True	True	True

$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

Baliokidetasun garrantzitsuak:

$\varphi \wedge True \equiv \varphi$	$\varphi \vee True \equiv True$	$\varphi \rightarrow True \equiv True$	$\varphi \leftrightarrow True \equiv \varphi$
$\varphi \wedge False \equiv False$	$\varphi \vee False \equiv \varphi$	$\varphi \rightarrow False \equiv \neg\varphi$	$\varphi \leftrightarrow False \equiv \neg\varphi$
		$False \rightarrow \varphi \equiv True$	
		$True \rightarrow \varphi \equiv \varphi$	

2.3.3.2. ∃ zenbatzaile existentzialaren semantika

∃x(φ) formularen esanahia φ formulak dioena betetzen duen elementuren bat (x deituko zaio elementuari) existitzen dela da. Gehiago zehaztuz, ∃x(φ) formulak dio gutxienez elementu batek φ formula betetzen duela.

Formula existentzialen ohiko formatua honako hau da:

$$\exists x(D(x) \wedge P(x))$$

Hor D(x) definizio-eremua edo tarte bat izango da (zenbaki osoen multzoko tarte bat adibidez) eta P(x) propietate bat izango da. Horrelako formula baten esanahia honako hau izango litzateke: D definizio-eremukoa edo tartekoa den eta P propietatea betetzen duen x elementuren bat existitzen da (gutxienez elementu bat existitzen dela ulertu behar da, baina ez dakigu zenbat diren).

6. adibidea:
Jarraian datorren formula existentzialaren bidez zenbaki osoz osatutako A(1..n) bektorean gutxienez elementu negatibo bat dagoela adierazten da:

$$\exists i(\underbrace{1 \le i \le n}_{D(i)} \wedge \underbrace{A(i) < 0}_{P(i)})$$

$\exists i(1 \le i \le n \wedge A(i) < 0)$ formulak honako hau dio:

$$A(1) < 0 \vee A(2) < 0 \vee ... \vee A(n) < 0$$

7. adibidea:
Jarraian datorren formula existentzialak x zenbakia 2ren berredura osoa dela dio:

$$\exists k(\underbrace{k \ge 0}_{D(k)} \wedge \underbrace{x = 2^k}_{P(k)})$$

$\exists k(k \ge 0 \wedge x = 2^k)$ formulak:

$$x = 2^0 \vee x = 2^1 \vee ... \vee x = 2^{50} \vee ...$$

8. adibidea:
Formula existentzial baten D **definizio-eremua hutsa denean**, formularen balioa **False** izango da, ez baita D definizio-eremukoa den eta P propietatea beteko duen elementurik existituko.

$$\exists z \underbrace{(1 \leq z \leq 0}_{D(z)} \wedge \underbrace{x = 2^z)}_{P(z)}$$

Ez da existitzen 1 baino handiagoa edo berdina eta 0 baino txikiagoa edo berdina den zenbaki osorik.

2.3.3.3. ∀ zenbatzaile unibertsalaren semantika

$\forall x(\varphi)$ formularen esanahia elementu denek (x deituko zaie elementuei) φ formulak dioena betetzen dutela da.

Formula unibertsalen ohiko formatua honako hau da:

$$\mathbf{\forall x(D(x) \rightarrow P(x))}$$

Hor D(x) definizio-eremu bat edo tarte bat izango da (zenbaki osoen multzoko tarte bat adibidez) eta P(x) propietate bat izango da. Horrelako formula baten esanahia honako hau izango litzateke: D definizio-eremuko edo tarteko elementu denek (x deituko zaie elementuei) P propietatea betetzen dute. Beste era batera esanda, elementu bat D definizio-eremukoa baldin bada orduan P propietatea beteko du.

9. adibidea:
Jarraian datorren formula unibertsalak zenbaki osozko A(1..n) bektoreko elementu denak positiboak direla adierazten du:

$$\forall i(\underbrace{1 \leq i \leq n}_{D(i)} \rightarrow \underbrace{A(i) > 0)}_{P(i)}$$

$\forall i(1 \leq i \leq n \rightarrow A(i) > 0)$ formulak honako hau dio:

$$A(1) > 0 \wedge A(2) > 0 \wedge ... \wedge A(n) > 0$$

10. adibidea:
Jarraian datorren formula unibertsalak zenbaki osozko A(1..n) bektorean x balioa ez dela agertzen adierazten du:

$$\forall i(\underbrace{1 \leq i \leq n}_{D(i)} \rightarrow \underbrace{A(i) \neq x}_{P(i)})$$

$\forall i(1 \leq i \leq n \rightarrow A(i) \neq x)$ formulak honako hau dio:

$$A(1) \neq x \wedge A(2) \neq x \wedge ... \wedge A(n) \neq x$$

11. adibidea:
Jarraian datorren formula unibertsalak zenbaki osozko A(1..n) bektoreko posizio bikoitietan x balioa ez dela agertzen adierazten du:

$$\forall i(\underbrace{(1 \leq i \leq n \wedge \text{bikoitia}(i))}_{D(i)} \rightarrow \underbrace{A(i) \neq x)}_{P(i)}$$

Kasu honetan bikoitia(i) predikatu bat da eta predikatu horrek True itzuliko du i bikoitia bada eta False bakoitia bada. Formula hori honela irakurriko litzateke: i edozein balio hartuta, i balioa 1 eta n-ren artean baldin badago eta bikoitia baldin bada, orduan A bektoreko i posizioko balioa x-ren desberdina izango da.

12. adibidea:
Formula unibertsal baten D **definizio-eremua hutsa denean**, formularen balioa **True** izango da, D definizio-eremuko elementu denek P propietatea beteko baitute. Beste era batera esanda, egia da elementu bat D definizio-eremukoa baldin bada, orduan P propietatea betetzen duela.

$$\forall i(\underbrace{1 < i \leq 1}_{D(i)} \rightarrow \underbrace{A(i) > 0)}_{P(i)}$$

Formula hori False izateko, 1 baino handiagoa den eta aldi berean 1 baino txikiagoa edo berdina den i balio bat aurkitu beharko genuke eta, gainera, i balio horrek A(i) > 0 ez luke bete beharko. Baina ezinezkoa da 1 baino handiagoa den eta aldi berean 1 baino txikiagoa edo berdina den i balio bat aurkitzea. Definizio-eremu horretan A(i) > 0 betetzen ez duen elementurik ezin denez aurkitu, formularen balioa True da.

2.3.3.4. Σ funtzioaren semantika

Σ funtzioa erabiltzeko hiru era ikusiko ditugu:

- $\sum_{i=m}^{n} \mathrm{f}(i)$ m eta n-ren artean dauden i elementuei dagozkien f(i) balioen batura da. Hor n-ren balioa m baino txikiagoa baldin bada, emaitza 0 izango da.

  Beraz, $\sum_{i=m}^{n} \mathrm{f}(i) = \mathrm{f}(m) + \mathrm{f}(m+1) + ... + \mathrm{f}(n)$

  13. adibidea:
  Jarraian datorren batukaria A bektoreko 2 eta 5 posizioen arteko elementuen batura da:

  $$\sum_{i=2}^{5} \mathrm{A}(i) = \mathrm{A}(2) + \mathrm{A}(3) + \mathrm{A}(4) + \mathrm{A}(5)$$

  14. adibidea:
  Batukariaren goiko muga behekoa baino txikiagoa denean emaitza 0 izango da:

  $$\sum_{k=2}^{7} k = 2+3+4+5+6+7 = 27 \qquad \sum_{k=7}^{2} k = 0$$

- $\sum_{i \in C} \mathrm{f}(i)$ C multzoan dauden i elementuei dagozkien f(i) balioen batura da. C multzoa hutsa baldin bada, emaitza 0 izango da.

  15. adibidea:
  C = {4, 8, 9} multzoa hartuz:

  $$\sum_{i \in C} x^i = x^4 + x^8 + x^9$$

  16. adibidea:
  Kontsideratutako multzoa hutsa (∅) denean batukariaren emaitza 0 izango da:

  $$\sum_{i \in \varnothing} x^i = 0$$

- $\sum_{P(i)} \mathrm{f}(i)$ P propietatea betetzen duten i elementuei dagozkien f(i) balioen batura da. P propietatea betetzen duen baliorik ez badago emaitza 0 izango da.

  17. adibidea:
  Kontsidera dezagun honako propietate hau: **P(i) ≡ 1 ≤ i ≤ 10 ∧ bikoitia(i)**

  $$\sum_{\mathrm{P(i)}} x^i = x^2 + x^4 + x^6 + x^8 + x^{10}$$

  P propietatea betetzen duten balioak 2, 4, 6, 8 eta 10 dira.

18. adibidea:
Adibide honetan kontsideratutako propietatea betetzen duen baliorik ez denez existitzen, emaitza 0 da. Propietatea honako hau da: **P(i) ≡ i ≥ 10 ∧ (8 mod i = 0)**. Propietate horren bidez 10 baino handiagoak edo berdinak diren eta 8 zenbakia (hondarrik sortzeke) zatitzen duten zenbakiak kontsideratzen ari gara baina propietate hori betetzen duen zenbakirik ez dago.

$$\sum_{\mathrm{P(i)}} x^i = 0$$

2.3.3.5. Π funtzioaren semantika

Π funtzioa erabiltzeko hiru era ikusiko ditugu:

- $\prod_{i=m}^{n} \mathrm{f}(i)$ m eta n-ren artean dauden i elementuei dagozkien f(i) balioen biderkadura da. Hor n-ren balioa m baino txikiagoa baldin bada, emaitza 1 izango da.
  Beraz, $\prod_{i=m}^{n} \mathrm{f}(i) = \mathrm{f}(m) * \mathrm{f}(m+1) * ... * \mathrm{f}(n)$

  19. adibidea:
  Jarraian datorren espresioa A bektoreko 2 eta 5 posizioen arteko elementuen biderkadura da:

  $$\prod_{i=2}^{5} \mathrm{A}(i) = \mathrm{A}(2) * \mathrm{A}(3) * \mathrm{A}(4) * \mathrm{A}(5)$$

  20. adibidea:
  Goiko muga behekoa baino txikiagoa denean emaitza 1 izango da:

  $$\prod_{k=2}^{7} k = 2 * 3 * 4 * 5 * 6 * 7 = 5040 \qquad \prod_{k=7}^{2} k = 1$$

- $\prod_{i \in C} \mathrm{f}(i)$ C multzoan dauden i elementuei dagozkien f(i) balioen biderkadura da. C multzoa hutsa baldin bada, emaitza 1 izango da.

  21. adibidea:
  C = {4, 8, 9} multzoa hartuz:

  $$\prod_{i \in C} x^i = x^4 * x^8 * x^9$$

  22. adibidea:
  Kontsideratutako multzoa hutsa (∅) denean emaitza 1 izango da:

  $$\prod_{i \in \varnothing} x^i = 1$$

- $\prod_{P(i)} f(i)$ P propietatea betetzen duten i elementuei dagozkien f(i) balioen biderkadura da. P propietatea betetzen duen baliorik ez badago emaitza 1 izango da.

  23. adibidea:
  Kontsidera dezagun honako propietate hau: **P(i) ≡ 1 ≤ i ≤ 10 ∧ bikoitia(i)**

  $$\prod_{P(i)} x^i = x^2 * x^4 * x^6 * x^8 * x^{10}$$

  P propietatea betetzen duten balioak 2, 4, 6, 8 eta 10 dira.

  24. adibidea:
  Adibide honetan kontsideratutako propietatea betetzen duen baliorik ez denez existitzen, emaitza 1 da. Propietatea honako hau da: **P(i) ≡ i ≥ 10 ∧ bikoitia(i) ∧ lehena(i)**. Propietate horretan lehena izeneko predikatuak True itzuliko du zenbakia lehena bada eta bestela False itzuliko du. Propietate horren bidez 10 baino handiagoak edo berdinak diren zenbaki bikoiti lehenak kontsideratzen ari gara baina propietate hori betetzen duen zenbakirik ez dago.

  $$\prod_{P(i)} x^i = 1$$

2.3.3.6. N funtzioaren semantika

N funtzioaren definizioa honako hau da:

$$Nx(D(x) \wedge P(x)) = kard\{x \mid D(x) \wedge P(x)\} = \sum_{D(x) \wedge P(x)} 1$$

Beraz Nx(D(x) ∧ P(x)) espresioak D definizio-eremukoak diren eta P propietatea betetzen duten zenbat elementu dauden adierazten digu. D eremua tarte bat edo multzo bat izan daiteke.

25. adibidea:
Ni(4 ≤ i ≤ 9 ∧ bikoitia(i)) = 3, izan ere 4 eta 9ren artean 3 elementu bikoiti daude (4, 6 eta 8).

26. adibidea:
Nx(x ∈ {3, 6, 14} ∧ bikoitia(x)) = 2, izan ere {3, 6, 14} multzoan bi elementu bikoiti daude (6 eta 14).

27. adibidea:
Nx(4 ≤ x ≤ 0 ∧ bikoitia(x)) = 0, ez baitago aldi berean 4 baino handiagoa edo berdina eta 0 baino txikiagoa edo berdina den elementurik.

28. adibidea:
Har dezagun jarraian daukagun A(1..8) bektorea:

3	15	6	7	20	14	11	33
1	2	3	4	5	6	7	8

$Nj(1 \leq j \leq 8 \wedge \text{bikoitia}(A(j))) = 3$ da bektore horretan hiru elementu bikoiti daudelako.

2.3.4. Aldagai askeak eta aldagai lotuak

Formula batean $\exists$ eta $\forall$ zenbatzaileekin erlazionatutako aldagaiei eta Σ, $\prod$ eta N funtzioekin erlazionatutako aldagaiei **aldagai lotuak** deitzen zaie eta beste aldagai denei **aldagai askeak** deitzen zaie.

Programa baten espezifikazioari dagokion formula bat idazterakoan edo esaldi bati dagokion formula bat idazterakoan honako bi arau hauek hartu behar dira kontuan:

- Programako edo esaldiko aldagaiak aldagai aske bezala agertzea komeni da.
- Formula idatzi ahal izateko asmatuko diren aldagai berriek beti aldagai lotu bezala agertu behar dute.

Beste era batera esanda, aldagai lotuak berriak izatea komeni da (programan edo esaldian agertzen ez diren aldagaiak) eta aldagai askeak programan edo esaldian agertzen diren aldagaiak izango dira.

29. adibidea:
"i eta j-ren artean (biak barne) ez dago x-en anizkoitzik den elementurik" esaldiari dagokion formula honako hau da:

$$\forall k(i \leq k \leq j \rightarrow k \bmod x \neq 0)$$

Formula horretan x, i eta j aldagaiak askeak dira. Konturatu justu esaldian agertzen diren aldagaiak direla. Bestalde, k aldagaia $\forall$ zenbatzaile unibertsalarekin doa eta horregatik lotua da. Konturatu k ez dela esaldian agertzen. Formula idatzi ahal izateko k aldagaia erabili da baina k ez da agertzen esaldian.

30. adibidea:
"Zenbaki osozko A(1..n) bektorean x behin bakarrik agertzen da"
Esaldiari dagokion formula honako hau da:

$$Ni(1 \leq i \leq n \wedge A(i) = x) = 1$$

Formula horretan A, n eta x aldagaiak askeak dira (justu esaldian agertzen direnak) eta i aldagaia N funtzioarekin doa eta lotua da. Konturatu i ez dela esaldian agertzen eta formula idazteko asmatu dugun aldagai berria dela.

Beste formula honek ere goiko esaldiak dioena adierazten du:

$$\exists i(1 \leq i \leq n \wedge x = A(i) \wedge \forall j((1 \leq j \leq n \wedge i \neq j) \rightarrow A(j) \neq x))$$

Kasu honetan i eta j aldagai lotuak dira, izan ere i aldagaia $\exists$ zenbatzaile existentzialarekin doa eta j aldagaia $\forall$ zenbatzaile unibertsalarekin doa.

2.3.5. Predikatu berrien definizioa

2.3.5.1. Predikatuen definizioa formulen bidez

Predikatuak formulen bidez definitzen dira. Predikatuaren argumentuak edo parametroak formulako aldagai askeak izango dira. Predikatu bat True edo False itzultzen duen funtzio bat dela esan dezakegu. Jarraian formulen bidez predikatuak nola definitu erakusten duten adibideak datoz. Gainera predikatuak definitu ondoren argumentu konkretuentzat predikatu horiek zein balio (True edo False) itzuliko luketen azaltzen da.

31. adibidea:
A(1..n) bektoreko elementu denak 1 baino handiagoak edo berdinak direla adierazten duen **denakpos(A(1..n))** predikatua honela definituko genuke

$$\text{denakpos}(A(1..n)) \equiv \forall i(1 \leq i \leq n \rightarrow A(i) \geq 1)$$

Kasu konkretuak:
denakpos((4, 10, 5, 8)) deiak True itzuliko luke.
denakpos((4, -10, 5, -8, 9)) deiak False itzuliko luke.

32. adibidea:
A(1..n) bektoreko elementu denak x baino handiagoak direla adierazten duen **denakhand(A(1..n), x)** predikatua honela definituko genuke:

$$\text{denakhand}(A(1..n), x) \equiv \forall i(1 \leq i \leq n \rightarrow A(i) > x)$$

Ikus daitekeen bezala, predikatuaren argumentu denak aldagai aske bezala agertzen dira formulan.

Kasu konkretuak:
denakhand((4, 10, 5, 8), 2) deiak True itzuliko luke.
denakhand((4, 10, 5, 8), 6) deiak False itzuliko luke.

33. adibidea:
A(1..n) bektorean gutxienez elementu bat x baino handiagoa dela adierazten duen **handiagorenbat(A(1..n), x)** predikatua honela definituko genuke:

$$\text{handiagorenbat}(A(1..n), x) \equiv \exists i(1 \leq i \leq n \wedge A(i) > x)$$

Kasu konkretuak:
handiagorenbat((4, 10, 5, 8), 6) deiak True itzuliko luke.
handiagorenbat((4, 10, 5, 8), 12) deiak False itzuliko luke.

34. adibidea:
A(1..n) bektoreko elementuen batura s dela adierazten duen **batura(A(1..n), s)** predikatua honela definituko genuke:

$$\text{batura}(A(1..n), s) \equiv \sum_{i=1}^{n} A(i) = s$$

Predikatuaren argumentu denak aldagai aske bezala agertzen dira formulan.

Kasu konkretuak:
batura((4, 12, 5, 8), 29) deiak True itzuliko luke.
batura((4, 12, 5, 8), 20) deiak False itzuliko luke.

35. adibidea:
Kasu honetan x-ren anizkoitzak diren A(1..n) bektoreko elementuen batura s dela adierazten duen **aniz_batu(A(1..n), s, x)** predikatuaren definizioa daukagu.

$$\text{aniz\_batu}(A(1..n), s, x) \equiv \sum_{1 \le i \le n \wedge A(i) \bmod x = 0} A(i) = s$$

Kasu konkretuak:
aniz_batu((4, 16, 5, 8), 28, 2) deiak True itzuliko luke.
aniz_batu((4, 16, 5, 8), 20, 2) deiak False itzuliko luke.
aniz_batu((4, 16, 5, 8), 0, 3) deiak True itzuliko luke.

36. adibidea:
Adibide honetan A(1..n) bektoreko elementu bakoitza (c_1, …, c_n) bektorean dagokion elementuaren bikoitza dela adierazten duen **bikoitza(A(1..n), (c_1, …, c_n))** predikatuaren definizioa daukagu:

$$\text{bikoitza}(A(1..n), (c_1, \ldots, c_n)) \equiv \forall i(1 \le i \le n \rightarrow A(i) = 2 * c_i)$$

Adibide honetan erakusten den bezala, bektore bat bi eratara adieraz daiteke, bere izena eta elementu-kopurua emanez A(1..n) taularen kasuan bezala edo taulako elementu denak emanez (c_1, …, c_n) taularen kasuan bezala.

Kasu konkretuak:
bikoitza((8, 32, 10, 16), (4, 16, 5, 8)) deiak True itzuliko luke.
bikoitza((8, 25, 10, 10), (4, 16, 5, 8)) deiak False itzuliko luke.
bikoitza((4, 16, 5, 8) , (8, 32, 10, 16)) deiak False itzuliko luke. Adibide honetan True itzultzeko, balio bikoitzek lehenengo bektorean egon beharko lukete eta ez bigarrenean.

2.3.5.2. Formula bati dagokion predikatua

Formula bat emanda, formula horri dagokion predikatua definitzeko izen bat asmatu beharko da eta parametro bezala formulako aldagai askeak ipiniko dira.

37. adibidea:
Formula: $\exists k(k \geq 0 \wedge x = 2^k)$
Dagokion predikatua: **biber(x)**

x balioa 2 zenbakiaren berredura osoa dela adierazten du formula horrek.

Kasu konkretuak:
biber(8) deiak True itzuliko luke $8 = 2^3$ delako.
biber(7) deiak False itzuliko luke 7 ez delako 2ren berredura osoa.

38. adibidea:
Formula: $Ni(1 \leq i \leq n \wedge A(i) = x) = 1$
Dagokion predikatua: **behin(A(1..n), x)**

A(1..n) bektorean x balioa behin bakarrik agertzen dela adierazten du formula horrek.

Kasu konkretuak:
behin((4, 10, 8, 6), 8) deiak True itzuliko luke.
behin((4, 10, 8, 6), 7) deiak False itzuliko luke.
behin((4, 10, 8, 6, 8), 8) deiak False itzuliko luke 8 behin baino gehiagotan agertzen delako.

39. adibidea:
Formula: $(Ni(1 \leq i \leq n \wedge A(i) = x) = 1) \wedge (Ni(1 \leq i \leq n \wedge A(i) = w) = 2)$
Dagokion predikatua: **bat_bi(A(1..n), x, w)**

A(1..n) bektorean x balioa behin agertzen dela eta w balioa bi aldiz agertzen dela adierazten du formula horrek.

Kasu konkretuak:
bat_bi((4, 10, 8, 6, 8), 6, 8) deiak True itzuliko luke 6 zenbakia behin agertzen delako eta 8 bi aldiz agertzen delako.
bat_bi((8, 10, 8, 6, 8), 6, 8) deiak False itzuliko luke, 6 zenbakia behin agertu arren 8 zenbakia ez delako justu bi aldiz agertzen.

2.3.5.3. Predikatu berriak nola definitu beste predikatu batzuk erabiliz

Aurretik definitu diren predikatuak ere erabil daitezke predikatu berriak definitzerakoan.

40. adibidea:
A(1..n) bektoreko elementu denak positiboak eta desberdinak direla adierazten duen **pos_desb(A(1..n))** predikatua definitu. Aurretik definituta dauden **denakpos** eta **behin** izeneko predikatuak erabili:

$$\text{pos\_desb}(A(1..n)) \equiv \text{denakpos}(A(1..n)) \wedge \forall j(1 \leq j \leq n \rightarrow \text{behin}(A(1..n), A(j)))$$

Kasu konkretuak:

Kontsidera ditzagun B(1..4) = (4, 16, 5, 8) eta C(1..5) = (4, 16, 8, 5, 8) bektoreak.

pos_desb(B(1..4)) deiak True itzuliko luke.
pos_desb(C(1..5)) deiak False itzuliko luke.
pos_desb((9, 5, -2, 7)) deiak False itzuliko luke.

2.4. Programen espezifikazioa: Adibideak

2.4.1. A(1..n) bektoreko elementuen batura s aldagaian kalkulatzen duen programa

A(1..n) bektoreko elementuen batura s aldagaian kalkulatzen duen honako programa honi dagokion aurre-ondoetako espezifikazioa emango da adibide honetan. Espezifikazioa ematerakoan $n \geq 1$ dela kontsideratu behar da.

```
{φ} ≡ ???
i : = 1;
s : = 0;
while i ≠ n + 1 loop
        s : = s + A(i);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:

$\{\varphi\} \equiv \{n \geq 1\}$

$\{\psi\} \equiv \{ s = \sum_{k=1}^{n} A(k) \}$

Aurre-ondoetako espezifikazioak sarrerako datuek bete behar dituzten baldintzak eta irteerako emaitzek bete behar dituzten baldintzak zein diren zehazteko balio du eta horregatik programan zehar laguntzaile bezala erabilitako aldagaiak ez dira aipatu behar. Adibide honetan i aldagaia bektorea zeharkatzeko indize bezala (laguntzaile bezala) erabiltzen da baina ez da sarrerako datu bat eta ez da emaitza bat, beraz espezifikazioan ez da agertzen. Programan agertzen diren aldagaietatik enuntziatuan agertzen direnak bakarrik aipatu beharko dira espezifikazioan. Baina gero programan agertzen ez diren aldagaiak erabil daitezke aldagai lotu bezala. Adibide honetan k-rekin hori gertatzen da.

2.4.2. B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa (1. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen honako programa honi dagokion aurre-ondoetako espezifikazioa emango da adibide honetan.

```
{φ} ≡ ???
i : = 0;
while i < n loop
        i : = i + 1;
        B(i) : = A(i);
end loop;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:
$\{\varphi\} \equiv \{n \geq 1\}$
$\{\psi\} \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k))\}$

Garrantzitsua da aurre-ondoetako espezifikazioan programan indize bezala erabiltzen den i aldagaia ez dela agertzen konturatzea. Aurre-ondoetako espezifikazioak sarrerako datuak eta irteerako emaitzak deskribatzeko balio du eta horregatik laguntzaile bezala erabilitako aldagaiak (i aldagaia kasu honetan) ez dira agertzen. Enuntziatuan bertan ere ez da i aldagaia agertzen eta horrek aurre-ondoetako espezifikazioan ez duela agertu behar adierazten digu.

2.4.3. B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa (2. bertsioa)

B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen beste programa honi dagokion aurre-ondoetako espezifikazioa emango da adibide honetan, $n \geq 1$ dela kontsideratuz.

```
{φ} ≡ ???
i : = n + 1;
while i > 1 loop
        i : = i – 1;
        B(i) : = A(i);
end loop;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:
$\{\varphi\} \equiv \{n \geq 1\}$
$\{\psi\} \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k))\}$

Aurreko adibideko programa eta programa hau desberdinak izan arren, egoera beretik abiatzen direnez eta emaitza bera lortzen dutenez, aurre-ondoetako espezifikazioa berdina da kasu bietan.

2.4.4. B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa (3. bertsioa)

B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen hirugarren programa honi dagokion aurre-ondoetako espezifikazioa emango da jarraian, $n \geq 1$ dela kontsideratuz.

```
{φ} ≡ ???
i : = 1;
while i ≤ n loop
        B(i) : = A(i);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:
$\{\varphi\} \equiv \{n \geq 1\}$
$\{\psi\} \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k))\}$

Adibide hau eta aurreko biak begiratuz, gauza bera egiten duten hiru programa desberdin ikus ditzakegu. Gauza bera egiten dutenez aurre-ondoetako espezifikazioa berdina da hiru kasuetan, espezifikazioak zer kalkulatuko den zehazten baitu eta ez nola kalkulatuko den.

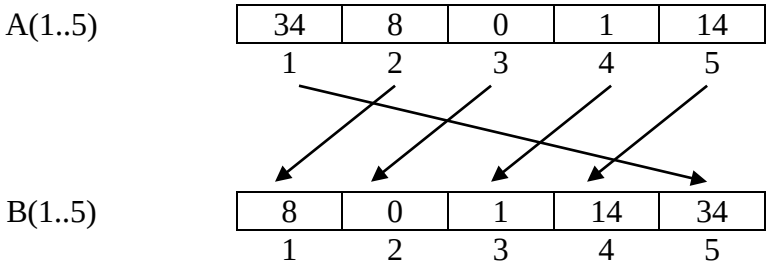
2.4.5. B(1..n) bektorean A(1..n) bektoreko elementuak posizio bat ezkerrera biratuta gordetzen dituen programa

Adibide honetan sarrera bezala gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea hartuta, A(1..n) bektoreko elementuak B(1..n) bektorean posizio bat ezkerrera biratuta gordetzen dituen honako programa honen aurre-ondoetako espezifikazioa emango da.

```
{φ} ≡ ???
B(n) := A(1);
i : = 1;
while i < n loop
        B(i) : = A(i + 1);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Adibidea

Adibide honen bidez kasu konkretu batean programak zer egingo lukeen erakusten da:



Aurre-ondoetako espezifikazioa:

$\{\varphi\} \equiv \{n \geq 1\}$

$\{\psi\} \equiv \{(B(n) = A(1)) \wedge \forall k(1 \leq k \leq n - 1 \rightarrow B(k) = A(k + 1))\}$

2.4.6. A(1..n) bektoreko elementuak A(1..n) bektorean bertan ezkerrera biratzen dituen programa

Kasu honetan sarrera bezala A(1..n) bektore bat emanda eta $n \geq 1$ dela jakinda, A(1..n) bektoreko elementuak ezkerrera biratzen dituen honako programa honen aurre-ondoetako espezifikazioa emango da.

```
{φ} ≡ ???
lag := A(1);
i : = 1;
while i < n loop
        A(i) : = A(i + 1);
        i : = i + 1;
end loop;
A(n) : = lag;
{ψ} ≡ ???
```

Hasteko **zuzena ez den** espezifikazio bat emango da:
$\{\varphi\} \equiv \{n \geq 1\}$
$\{\psi\} \equiv \{(A(n) = A(1)) \wedge \forall k(1 \leq k \leq n - 1 \rightarrow A(k) = A(k + 1))\}$

Jarraian datorren adibide konkretuak goiko espezifikazioa ez dela zuzena erakusten du:

A(1..5) bektorea programa exekutatu aurretik	34	8	0	1	14
	1	2	3	4	5
A(1..5) bektorea programa exekutatu ondoren	8	0	1	14	34
	1	2	3	4	5

Hor ikusten den bezala, bukaerako baldintza ez da betetzen, ez da egia A(5) = A(1) denik eta ezta [1..n – 1] tarteko elementuentzat A(k) = A(k + 1) betetzen denik ere.

Espezifikazioaren bidez hasieran A(1) posizioan zegoen balioa bukaeran A(n) posizioan egongo dela eta [1..n – 1] tarteko k posizio bakoitzean hasieran k + 1 posizioan zegoena egongo dela esan nahi da. Baina prozesu horretan A(1..n) bektorea aldatu egiten denez, A(1..n) bektoreko hasierako balioak erreferentziatzeko aurrebaldintzan izen bat eman beharko zaie. Programa exekutatu aurretik k posizioan zegoen elementuari a_k deituko diogu:

Aurre-ondoetako espezifikazioa:
$\{\varphi\} \equiv \{n \geq 1 \wedge \forall k(1 \leq k \leq n \rightarrow A(k) = a_k)\}$
$\{\psi\} \equiv \{(A(n) = a_1) \wedge \forall k(1 \leq k \leq n - 1 \rightarrow A(k) = a_{k+1})\}$

Garrantzitsua da aurre-ondoetako espezifikazioan lag eta i izeneko aldagaiak ez direla agertzen konturatzea. Aurre-ondoetako espezifikazioaren bidez sarrerako datuak eta irteerako emaitzak deskribatzen dira eta horregatik bitartean laguntzaile bezala erabilitako aldagaiak ez dira aipatzen ez baitira ez datuak eta ez emaitzak.

2.4.7. c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen programa (1. bertsioa)

Kasu honetan sarrera bezala gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen honako programa honi dagokion aurre-ondoetako espezifikazioa emango da.

```
{φ} ≡ ???
i : = 1;
c : = 0;
while i <= n loop
        if A(i) = 0 then c : = c + 1; end if;
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:
$\{\varphi\} \equiv \{n \geq 1\}$
$\{\psi\} \equiv \{c = Nk(1 \leq k \leq n \wedge A(k) = 0)\}$

2.4.8. c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen programa (2. bertsioa)

Kasu honetan sarrera bezala gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen honako beste programa honi dagokion aurre-ondoetako espezifikazioa emango da.

```
{φ} ≡ ???
i : = 0;
c : = 0;
while i < n loop
        i : = i + 1;
        if A(i) = 0 then c : = c + 1; end if;
end loop;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:
$\{\varphi\} \equiv \{n \geq 1\}$
$\{\psi\} \equiv \{c = Nk(1 \leq k \leq n \wedge A(k) = 0)\}$

Adibide hau eta aurrekoa begiratuz, programak desberdinak izan arren gauza bera egiten dutenez espezifikazioa berdina dela ikus dezakegu.

2.4.9. b aldagaian A(1..n) bektorean zerorik agertzen al den ala ez erabakitzen duen programa

Kasu honetan sarrera bezala gutxienez elementu bat (n ≥ 1) duen A(1..n) bektorea emanda, b aldagai boolearrean A(1..n) bektorean zerorik ba al dagoen ala ez erabakitzen duen honako programa honi dagokion aurre-ondoetako espezifikazioa emango da:

```
{φ} ≡ ???
i : = 1;
b : = False;
while i <= n and not b loop
        b : = (A(i) = 0);
        i : = i + 1;
end loop;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:
{φ} ≡ {n ≥ 1}
{ψ} ≡ {b ↔ ∃k(1 ≤ k ≤ n ∧ A(k) = 0)}

2.4.10. b aldagaian A(1..n) bektoreko bikoiti kopurua bakoiti kopurua baino handiagoa al den erabakitzen duen programa

Sarrera bezala A(1..n) bektore bat emanda eta n ≥ 1 dela jakinda, b aldagai boolearrean A(1..n) bektorean elementu bikoitien kopurua bakoitien kopurua baino handiagoa al den erabakitzen duen honako programa honi dagokion aurre-ondoetako espezifikazioa emango da:

```
{φ} ≡ ???
i : = 1;
bik: = 0; bak := 0;
while i <= n loop
        if A(i) mod 2 = 0 then bik : = bik + 1;
        else bak : = bak + 1; end if;
        i : = i + 1;
end loop;
b : = (bik > bak);
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:
{φ} ≡ {n ≥ 1}
{ψ} ≡ {b ↔ Nk(1 ≤ k ≤ n ∧ bikoitia(A(k))) > Nk(1 ≤ k ≤ n ∧ ¬bikoitia(A(k)))}

Konturatu emaitza partzialak lortzeko erabili diren aldagaiak eta laguntzaile gisa erabili diren aldagaiak ez direla agertzen aurre-ondoetako espezifikazioan. Adibide honetan bik, bak eta i dira era horretako aldagaiak. Sarrerako datua A(1..n) da eta irteerakoa b da. Beraz beste hiru aldagaiak ez dira agertuko aurre-ondoetako espezifikazioan, aurre-ondoetako espezifikazioan sarrerako datuak eta irteerako emaitzak bakarrik deskribatzen baitira.

2.4.11. x balioa A(1..n) bektorean agertzen dela jakinda, bere lehenengo agerpenaren posizioa pos aldagaian kalkulatzen duen programa

x balioa A(1..n) bektorean agertzen dela jakinda, bere lehenengo agerpenaren posizioa pos aldagaian kalkulatzen duen honako programa honen aurre-ondoetako espezifikazioa emango da jarraian. Espezifikazioa ematerakoan $n \geq 1$ betetzen dela kontsideratu behar da.

```
{φ} ≡ ???
i : = 1;
segi: = (A(1) ≠ x);
while segi loop
        segi: = (A(i + 1) ≠ x);
        i : = i + 1;
end loop;
pos : = i;
{ψ} ≡ ???
```

Aurre-ondoetako espezifikazioa:

$\{\varphi\} \equiv \{n \geq 1 \wedge \exists k(1 \leq k \leq n \wedge A(k) = x)\}$

$\{\psi\} \equiv \{1 \leq pos \leq n \wedge A(pos) = x \wedge \forall k(1 \leq k \leq pos - 1 \rightarrow A(k) \neq x)\}$

φ aurreko baldintzaren edo hasierako baldintzaren bidez sarrerako bektoreak gutxienez elementu bat izango duela eta bektore horretan x gutxienez behin agertuko dela adierazten da. Bestalde, programaren bukaerako baldintzaren bidez (ψ-ren bidez) bukaeran pos aldagaian x balioaren lehenengo agerpenari dagokion posizioa edukiko dela adierazten da.

2.4.12. x balioa A(1..n) bektorean agertzen bada pos-en bere lehenengo agerpenaren posizioa eta bestela n + 1 balioa itzultzen duen programa

x balioa A(1..n) bektorean agertzen bada, pos aldagaian x-ren lehenengo agerpenaren posizioa eta bestela n + 1 balioa itzultzen duen honako programa honen aurre-ondoetako espezifikazioa emango da jarraian. Espezifikazioa ematean $n \geq 1$ betetzen dela kontsideratu.

```
{φ} ≡ ???
i : = 2;
segi : = (A(1) ≠ x);
while i ≤ n ∧ segi loop
        segi : = (A(i) ≠ x);
        i : = i + 1;
end loop;
if segi then pos : = i;
else pos : = i – 1;
end if;
{ψ} ≡ ???
```

Jarraian adierazten den eran definitutako **badago(z, C(1..r))** predikatua erabiliko da:

$$\exists k(1 \leq k \leq r \wedge C(k) = z)$$

Aurre-ondoetako espezifikazioa:

$\{\varphi\} \equiv \{n \geq 1\}$

$\{\psi\} \equiv \{1 \leq pos \leq n + 1 \wedge \neg badago(x, A(1..pos – 1)) \wedge$
$badago(x, A(1..n)) \rightarrow (1 \leq pos \leq n \wedge A(pos) = x) \wedge$
$\neg badago(x, A(1..n)) \rightarrow pos = n + 1\}$

φ aurrebaldintzaren bidez sarrerako bektoreak gutxienez elementu bat izango duela adierazten da. Bestalde, programaren ondorengo (edo bukaerako) baldintzaren bidez (ψ-ren bidez) bukaeran x agertu bada, pos aldagaian x balioaren lehenengo agerpenari dagokion posizioa edukiko dela adierazten da eta x ez bada agertu, pos aldagaian n + 1 balioa edukiko dela adierazten da.

2.5. Programen dokumentazioa: jarraitu beharreko ideia

Programen dokumentazioari buruzko ariketetan programa bat emango zaigu eta programan zehaztutako asertzioak idatzi beharko dira.
Asertzio bat programako puntu batean betetzen den propietatea adierazten duen formula bat izango da.
Honako urratsak jarraitu beharko dira:

- **Programa ez iteratiboetan**, lehenengo hasierako baldintza eman eta, gero, baldintza horretatik abiatu eta exekutatzen diren aginduek eragiten dituzten aldaketak kontuan hartuz, tarteko asertzioak eta bukaerako baldintza lortu beharko dira.

- Programa iteratiboetan, lehenengo urratsa hasierako baldintza, bukaerako baldintza eta inbariantea ematea izango da. Eta gero, while-aren barruko asertzioak lortu beharko dira. Horretarako, inbariantetik abiatu eta while-aren baldintzaren ebaluazioak eta while-aren barruan exekutatzen diren aginduek eragiten dituzten aldaketak kontuan hartu beharko dira. Beraz, inbariantean aldaketak eginez lortuko dira while-aren barruko asertzioak.

- Programa iteratiboetan while-ak gehienez zenbat buelta emango dituen adierazten duen E espresioa ere eskatuko da. Justu inbariantea betetzen den puntuan gaudenean E espresioak gehienez zenbat buelta falta diren adieraziko du. Batzuetan E espresioak while-ari justu zenbat buelta gelditzen zaizkion adieraziko du era zehatzean eta beste batzuetan, aldiz, gehienez zenbat buelta gelditzen zaizkion adieraziko du.

Gehienetan programa dokumentatzen hasi aurretik programa dokumentatzeko erabiliko diren predikatuak definitzea eskatuko da. Predikatuak erabiliz formulak laburragoak eta ulertzen errazagoak izatea lortzen da (programazioan funtzioak eta prozedurak erabiliz programa argiagoak idaztea lortzen den bezala).

Jarraian programak nola dokumentatu behar diren erakusten duten adibide batzuk datoz.

2.6. Programen dokumentazioa: Adibideak

2.6.1. B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordetzen dituen programa (1. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordeko dituen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez. Bektorea ezkerretik eskuinera zeharkatzen da.

```
(1) ≡ {Hasierako baldintza}
i : = 1;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i ≠ n + 1 loop
        (4) ≡ {Tarteko asertzioa}
        B(i) : = A(i) * x;
        (5) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (6) ≡ {Tarteko asertzioa}
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4), (5) eta (6) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

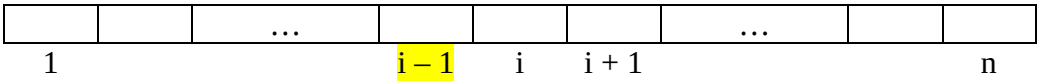
- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k) * x)\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (7) puntuko formularen bidez, amaieran B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta edukiko ditugula adierazten da.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 1 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean, edozein bueltatan, egoera zein den deskribatuko da. Beraz, i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz, inbariantean, hau da, programako (3) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioko balioa B-ko i posizioan x balioaz biderkatuta gordetzea denez, (3) puntuan gaudenean A(1..n) bektoreko i posizioko elementua bider x oraindik B bektorean gordetzeke dagoela baiezta dezakegu. Gorde den azkena i – 1 posiziokoa da.

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa ez dela n + 1 eta hori da, hain zuzen ere, (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta B taulako i posizioan A(i) * x gorde dela kontuan hartuz kalkulatuko da. Baina (5) puntuan i-ri oraindik ez zaio 1 gehitu. Beraz, kopiatutako elementuak 1 posiziotik i posiziora artekoak izango dira.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i-ren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Beraz, kalkulatutako elementuak 1 posiziotik i – 1 posiziora artekoak izango dira berriro ere. Eta i-ren balioa gutxienez 2 izango da eta gehienez n + 1.

  $(6) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- Amaitzeko, E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz, kasu honetan E = n + 1 – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori

gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.2. B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordetzen dituen programa (2. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordeko dituen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez. Bektorea ezkerretik eskuinera zeharkatzen da. Programa honek eta 2.6.1 atalean aurkeztutakoak emaitza bera kalkulatzen dute baina era desberdinean. Emaitzak kalkulatzeko eran dauden desberdintasun horiek dokumentazioan duten eragina zein den ikusi nahi da.

```
(1) ≡ {Hasierako baldintza}
i : = 1;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i ≠ n + 1 loop
        (4) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (5) ≡ {Tarteko asertzioa}
        B(i – 1) : = A(i – 1) * x;
        (6) ≡ {Tarteko asertzioa}
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko, hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4), (5) eta (6) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko, E espresioa eman beharko da:

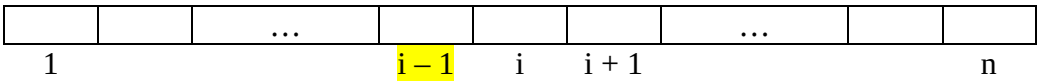
- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k) * x)\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (7) puntuko formularen bidez, amaieran B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta edukiko ditugula adierazten da. Bi formula hauek 2.6.1 ataleko (1) eta (7) formulen berdinak dira programa biak egoera beretik abiatzen direlako eta biek emaitza bera lortzen dutelako.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 1 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean, edozein bueltatan, egoera zein den deskribatuko da. Beraz, i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz, inbariantean, hau da, programako (3) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren i-ren balioari 1 gehitzen zaionez eta gero aurreko posizioari dagokion kalkulua burutzen denez, (3) puntuan gaudenean A(1..n) bektoreko i posizioko elementua bider x oraindik B bektorean gordetzeke dagoela baiezta dezakegu. Gorde den azkena i – 1 posiziokoa da.

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa ez dela n + 1 eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i-ren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Baina (5) oraindik A(i) * x ez da B(i) posizioan gorde. Beraz, kalkulatutako elementuak 1 posiziotik i – 2 posiziora artekoak izango dira berriro ere.

  $(5) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 2 \rightarrow B(k) = A(k) * x)\}$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta B taulako i – 1 posizioan A(i – 1) * x gorde dela kontuan hartuz kalkulatuko da. Beraz, kopiatutako elementuak 1 posiziotik i – 1 posiziora artekoak izango dira.

  $(6) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k) * x)\}$

- Amaitzeko, E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz, kasu honetan E = n + 1 – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori

gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.3. B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordetzen dituen programa (3. bertsioa)

Gutxienez elementu bat (n ≥ 1) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordeko dituen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez. Bektorea ezkerretik eskuinera zeharkatzen da. Programa honek eta 2.6.1 eta 2.6.2 ataletan aurkeztutakoek emaitza bera kalkulatzen dute baina era desberdinean. Helburua gauzak era desberdinean egiteak dokumentazioan duen eragina zein den ikustea da.

```
(1) ≡ {Hasierako baldintza}
i : = 0;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i ≠ n loop
        (4) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (5) ≡ {Tarteko asertzioa}
        B(i) : = A(i) * x;
        (6) ≡ {Tarteko asertzioa}
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4), (5) eta (6) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

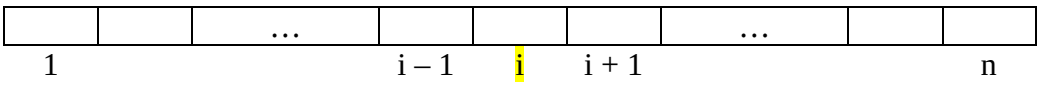
- (1) ≡ {n ≥ 1}
  (7) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k) * x)}

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (7) puntuko formularen bidez, amaieran B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta edukiko ditugula adierazten da. Bi formula hauek 2.6.1 eta 2.6.2 ataletakoko (1) eta (7) formulen berdinak dira programa hauek egoera beretik abiatzen direlako eta emaitza bera lortzen dutelako.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 0 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 0 balio duela baiezta dezakegu:

(2) ≡ {n ≥ 1 ∧ i = 0}

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak 0 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 0 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza i-ri 1 balioa gehitzea denez, (3) puntuan gaudenean A(1..n) bektoreko i posizioko elementua bider x dagoeneko B bektorean gordeta dagoela baiezta dezakegu.

(3) ≡ {(0 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i → B(k) = A(k) * x)}

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa ez dela n eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

(4) ≡ {(0 ≤ i ≤ n – 1) ∧ ∀k(1 ≤ k ≤ i → B(k) = A(k) * x)}

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i-ren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Baina (5) puntuan oraindik B taulako i posizioan A(i) * x ez da gorde. Beraz orain i ez da 0 izango eta kalkulatutako elementuak 1 posiziotik i – 1 posiziora artekoak izango dira.

(5) ≡ {(1 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i – 1 → B(k) = A(k) * x)}

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta B taulako i posizioan A(i) * x gorde dela kontuan hartuz kalkulatuko da. Beraz kopiatutako elementuak 1 posiziotik i posiziora artekoak izango dira.

(6) ≡ {(1 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i → B(k) = A(k) * x)}

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori

gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.4. B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordetzen dituen programa (4. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta gordeko dituen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez. Bektorea ezkerretik eskuinera zeharkatzen da. Programa honek eta 2.6.1, 2.6.2 eta 2.6.3 ataletan aurkeztutakoek emaitza bera kalkulatzen dute baina era desberdinean. Helburua gauzak era desberdinean egiteak dokumentazioan duen eragina zein den ikustea da.

```
(1) ≡ {Hasierako baldintza}
i : = 0;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i ≠ n loop
        (4) ≡ {Tarteko asertzioa}
        B(i + 1) : = A(i + 1) * x;
        (5) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (6) ≡ {Tarteko asertzioa}
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4), (5) eta (6) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k) * x)\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (7) puntuko formularen bidez, amaieran B(1..n) bektorean A(1..n) bektoreko elementuak x balioaz biderkatuta edukiko ditugula adierazten da. Bi formula hauek 2.6.1, 2.6.2 eta 2.6.3 ataletako (1) eta (7) formulen berdinak dira programa hauek egoera beretik abiatzen direlako eta emaitza bera lortzen dutelako

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 0 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 0 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 0\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.

  Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak 0 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 0 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i + 1 posizioko balioa B-ko i + 1 posizioan x balioaz biderkatuta gordetzea denez, (3) puntuan gaudenean A(1..n) bektoreko i posizioko elementua bider x dagoeneko B bektorean gordeta dagoela baiezta dezakegu.

  $(3) \equiv \{(0 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa ez dela n eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

  $(4) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta B taulako i + 1 posizioan A(i + 1) * x gorde dela kontuan hartuz kalkulatuko da. Baina (5) puntuan i-ri oraindik ez zaio 1 gehitu. Beraz kopiatutako elementuak 1 posiziotik i + 1 posiziora artekoak izango dira.

  $(5) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i + 1 \rightarrow B(k) = A(k) * x)\}$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i-ren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Beraz kalkulatutako elementuak 1 posiziotik i posiziora artekoak izango dira. Orain i-ren balioa 1 eta n-ren artean egongo dela ziurta dezakegu.

  $(6) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k) * x)\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko

programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.5. A(1..n) bektoreko elementuak x balioaz biderkatzen dituen programa

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, A(1..n) bektoreko elementuak x balioaz biderkatuko dituen honako programa hau dokumentatu hor zehazten diren formulak emanez. Bektorea ezkerretik eskuinera zeharkatzen da. Aurreko lau adibideetan ez bezala, adibide honetan aldaketak A bektorean bertan egiten dira.

```
(1) ≡ {Hasierako baldintza}
i : = 0;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i ≠ n loop
        (4) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (5) ≡ {Tarteko asertzioa}
        A(i) : = A(i) * x;
        (6) ≡ {Tarteko asertzioa}
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4), (5) eta (6) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

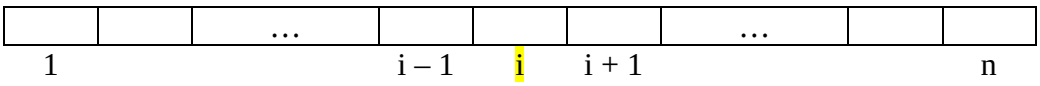
- A(1..n) bektoreetako posizioetako balioak aldatu egingo direnez eta balio berriak lehendik zeudenen bidez kalkulatuko direnez, hasierako balioei izena eman behar zaie hasierako baldintzan. Hasierako balioei izena emateko normalean minuskulak erabiltzen dira.
  $(1) \equiv \{n \geq 1 \wedge \forall k(1 \leq k \leq n \rightarrow A(k) = a_k)\}$
  $(7) \equiv \{\forall k(1 \leq k \leq n \rightarrow A(k) = a_k * x)\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela eta A bektoreko k posizio bakoitzean hasieran a_k balioa daukagula adierazten da. Bestalde (7) puntuko formularen bidez, amaieran A(1..n) bektoreko k posizio bakoitzean hasierako a_k balioa bider x edukiko dela adierazten da.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 0 balioa esleitu zaiola kontuan hartu beharko da. Beraz (2) puntuan (1) puntuko dena eta i = 0 beteko da:

(2) ≡ {n ≥ 1 ∧ ∀k(1 ≤ k ≤ n → A(k) = a_k) ∧ i = 0}

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak 0 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 0 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza i-ri 1 balioa gehitzea denez, (3) puntuan gaudenean A(1..n) bektoreko i posizioko elementua bider x dagoeneko A bektorean gordeta dagoela baiezta dezakegu.

(3) ≡ {(0 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i → A(k) = a_k * x)}

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa ez dela n eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

(4) ≡ {(0 ≤ i ≤ n – 1) ∧ ∀k(1 ≤ k ≤ i → A(k) = a_k * x)}

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i-ren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Baina (5) puntuan oraindik A taulako i posizioan a_i * x ez da gorde. Beraz orain i ez da 0 izango eta kalkulatutako elementuak 1 posiziotik i – 1 posiziora artekoak izango dira.

(5) ≡ {(1 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i – 1 → A(k) = a_k * x)}

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta A taulako i posizioan a_i * x gorde dela kontuan hartuz kalkulatuko da. Beraz kopiatutako elementuak 1 posiziotik i posiziora artekoak izango dira.

(6) ≡ {(1 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i → A(k) = a_k * x)}

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurreondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori

gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.6. A(1..n) bektoreko elementuen batura s aldagaian kalkulatzen duen programa (1. bertsioa)

Gutxienez osagai bat izango duen A(1..n) bektoreko elementuen batura s aldagaian kalkulatzen duen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez.

```
(1) ≡ {Hasierako baldintza}
i : = 0;
(2) ≡ {Tarteko asertzioa}
s : = 0;
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariante} i < n loop
        (5) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (6) ≡ {Tarteko asertzioa}
        s : = s + A(i);
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. Bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (5) eta (6) puntuetako tarteko asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

- $(1) \equiv \{n \geq 1\}$

  $(7) \equiv \{ s = \sum_{k=1}^{n} A(k) \}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (6) puntuko formularen bidez, amaieran s aldagaian A(1..n) bektoreko elementuen batura edukiko dela adierazten da.

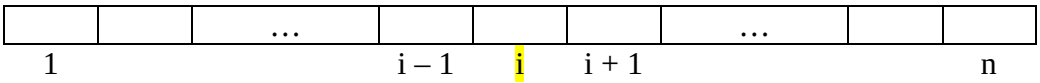
- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 0 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 0 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 0\}$

- (3) puntuko asertzioa (2) puntuko tarteko asertziotik abiatuta eta s aldagaiari 0 balioa esleitu zaiola kontuan hartuz kalkulatuko da. (3) puntuan n 1 baino handiagoa edo berdina dela, i aldagaiak 0 balio duela eta s-ren balioa 0 dela baiezta dezakegu:

  $(3) \equiv \{n \geq 1 \wedge i = 0 \wedge s = 0\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 0 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 0 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza i-ri 1 gehitzea denez, (4) puntuan gaudenean A(1..n) bektoreko i posizioko elementua s-ri dagoeneko batu zaiola baiezta dezakegu.

$$(4) \equiv \{(0 \leq i \leq n) \wedge s = \sum_{k=1}^{i} A(k)\}$$

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n baino txikiagoa dela eta hori da hain zuzen ere (5) puntuaren eta (4) puntuaren arteko desberdintasuna.

$$(5) \equiv \{(0 \leq i \leq n - 1) \wedge s = \sum_{k=1}^{i} A(k)\}$$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i aldagaiaren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Baina (6) puntuan i-ren balio berriak adierazten duen posizioko elementua oraindik ez zaio s-ri batu. Beraz, i-ren balioari 1 gehitu ondoren i-ren balioa 0 ez dela izango eta orain n ere izan litekeela baiezta dezakegu. Bestalde, s aldagaian batutakoak 1 posiziotik i – 1 posiziora artekoak izango dira.

$$(6) \equiv \{(1 \leq i \leq n) \wedge s = \sum_{k=1}^{i-1} A(k)\}$$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari

garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.7. A(1..n) bektoreko elementuen batura s aldagaian kalkulatzen duen programa (2. bertsioa)

Gutxienez osagai bat izango duen A(1..n) bektoreko elementuen batura s aldagaian kalkulatzen duen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez.

```
(1) ≡ {Hasierako baldintza}
i : = 1;
(2) ≡ {Tarteko asertzioa}
s : = 0;
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariantea} i ≤ n loop
        (5) ≡ {Tarteko asertzioa}
        s : = s + A(i);
        (6) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. Bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (5) eta (6) puntuetako tarteko asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da.

Aurreko adibideko programa eta programa hau egoera beretik abiatzen direnez eta emaitza bera kalkulatzen dutenez, kasu bietan hasierako eta bukaerako baldintzak berdinak dira. Baina emaitza kalkulatzeko era desberdina denez bi programetan, inbariantea eta tarteko asertzioak ere desberdinak dira, antzekoak baina desberdinak programak ere antzekoak baina desberdinak diren neurri berean.

- $(1) \equiv \{n \geq 1\}$

  $(7) \equiv \{ s = \sum_{k=1}^{n} A(k) \}$

(1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (6) puntuko formularen bidez, amaieran s aldagaian A(1..n) bektoreko elementuen batura edukiko dela adierazten da.

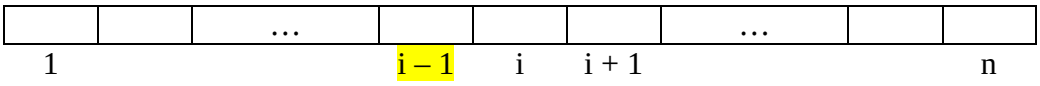
- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 1 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- (3) puntuko asertzioa (2) puntuko asertziotik abiatuta eta s aldagaiari 0 balioa esleitu zaiola kontuan hartuz kalkulatuko da. (3) puntuan n 1 baino handiagoa edo berdina dela, i aldagaiak 1 balio duela eta s-ren balioa 0 dela baiezta dezakegu:

  $(3) \equiv \{n \geq 1 \wedge i = 1 \wedge s = 0\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza s aldagaiari A bektoreko i posizioko balioa gehitzea denez, (4) puntuan gaudenean A(1..n) bektoreko i posizioko elementua oraindik s-ri ez zaiola batu baiezta dezakegu. Batu den azkena i – 1 posiziokoa da.

$$(4) \equiv \{(1 \leq i \leq n + 1) \wedge s = \sum_{k=1}^{i-1} A(k)\}$$

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n + 1 baino txikiagoa dela eta hori da hain zuzen ere (5) puntuaren eta (4) puntuaren arteko desberdintasuna.

$$(5) \equiv \{(1 \leq i \leq n) \wedge s = \sum_{k=1}^{i-1} A(k)\}$$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i posizioko elementua s-ri batu zaiola kontuan hartuz kalkulatuko da. Baina (6) puntuan i-ri oraindik ez zaio 1 gehitu. Beraz, s aldagaian batutakoak 1 posiziotik i posiziora artekoak izango dira.

$$(6) \equiv \{(1 \leq i \leq n) \wedge s = \sum_{k=1}^{i} A(k)\}$$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n + 1 – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.8. B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa(ezkerretik eskuinera zeharkatuz) (1. bertsioa)

Gutxienez elementu bat (n ≥ 1) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen honako programa hau dokumentatu hor zehazten diren formulak emanez. Bektorea ezkerretik eskuinera zeharkazten da.

```
(1) ≡ {Hasierako baldintza}
i : = 0;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i < n loop
        (4) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (5) ≡ {Tarteko asertzioa}
        B(i) : = A(i);
end loop;
(6) ≡ {Bukaerako baldintza}
(7) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4) eta (5) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

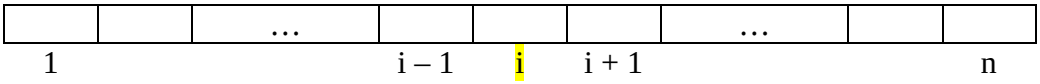
- (1) ≡ {n ≥ 1}
  (6) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k))}

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (6) puntuko formularen bidez, amaieran A(1..n) eta B(1..n) bektoreetan elementu berdinak edukiko ditugula adierazten da.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 0 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 0 balio duela baiezta dezakegu:

  (2) ≡ {n ≥ 1 ∧ i = 0}

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak 0 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 0 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza i-ri 1 gehitzea denez, (3) puntuan gaudenean A(1..n) bektoreko i posizioko elementua A-tik B-ra kopiatuta dago:

  $(3) \equiv \{(0 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k))\}$

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n baino txikiagoa dela eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

  $(4) \equiv \{(0 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k))\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i aldagaiaren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Baina (5) puntuan i-ren balio berriak adierazten duen posizioko A-ko elementua oraindik ez da B-ra kopiatu. Beraz, i-ren balioari 1 gehitu ondoren i-ren balioa 0 ez dela izango eta orain n ere izan litekeela baiezta dezakegu. Bestalde, B-ra kopiatutakoak 1 posiziotik i – 1 posiziora artekoak izango dira.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k))\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.9. B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa (ezkerretik eskuinera zeharkatuz) (2. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen honako programa hau dokumentatu hor zehazten diren formulak emanez. Bektorea ezkerretik eskuinera zeharkatzen da eta programa hau 2.6.8 atalean aurkeztutakoaren antzekoa izan arren, ez da berdina eta desberdintasun horrek dokumentazioan duen eragina zein den ikusi nahi da. Programa biek emaitza bera kalkulatzen dute baina era desberdinean.

```
(1) ≡ {Hasierako baldintza}
i : = 1;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i ≤ n loop
        (4) ≡ {Tarteko asertzioa}
        B(i) : = A(i);
        (5) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(6) ≡ {Bukaerako baldintza}
(7) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4) eta (5) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

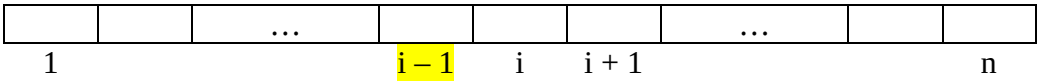
- $(1) \equiv \{n \geq 1\}$
  $(6) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k))\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (6) puntuko formularen bidez, amaieran A(1..n) eta B(1..n) bektoreetan elementu berdinak edukiko ditugula adierazten da.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 1 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioko balioa B-ko i posiziora kopiatzea denez, (3) puntuan gaudenean A(1..n) bektoreko i posizioko elementua oraindik B-ra kopiatzeke dagoela baiezta dezakegu. kopiatu den azkena i – 1 posiziokoa da.

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k))\}$

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n + 1 baino txikiagoa dela eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k))\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i posizioko elementua A-tik B-ra kopiatua izan dela kontuan hartuz kalkulatuko da. Baina (5) puntuan i-ri oraindik ez zaio 1 gehitu. Beraz kopiatutako elementuak 1 posiziotik i posiziora artekoak izango dira.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k))\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n + 1 – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.10. B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa (eskuinetik ezkerrera zeharkatuz) (1. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez. Bektorea eskuinetik ezkerrera zeharkatzen da eta programa hau 2.6.8 eta 2.6.9 ataletan aurkeztutakoen antzekoa izan arren, ez da berdina eta desberdintasun horrek dokumentazioan duen eragina zein den ikusi nahi da. Hiru programek emaitza bera kalkulatzen dute baina era desberdinean.

```
(1) ≡ {Hasierako baldintza}
i : = n + 1;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i > 1 loop
        (4) ≡ {Tarteko asertzioa}
        i : = i – 1;
        (5) ≡ {Tarteko asertzioa}
        B(i) : = A(i);
end loop;
(6) ≡ {Bukaerako baldintza}
(7) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4) eta (5) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

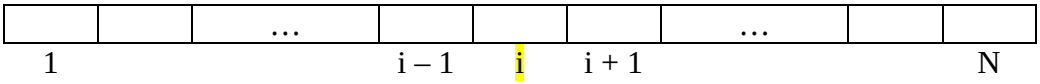
- $(1) \equiv \{n \geq 1\}$
  $(6) \equiv \{\forall k(1 \leq k \leq n \rightarrow B(k) = A(k))\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (6) puntuko formularen bidez, amaieran A(1..n) eta B(1..n) bektoreetan elementu berdinak edukiko ditugula adierazten da. Hasierako baldintza eta bukaerako baldintza 2.6.8 eta 2.6.9 ataletan emandakoen berdinak dira.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari n + 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak n + 1 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = n + 1\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaiak bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak n + 1 eta 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian n + 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza i-ri 1 kentzea denez, (3) puntuan gaudenean A(1..n) bektoreko i eta n posizioen arteko elementuak A-tik B-ra kopiatuta daude.

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(i \leq k \leq n \rightarrow B(k) = A(k))\}$

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa 1 baino handiagoa dela eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

  $(4) \equiv \{(2 \leq i \leq n + 1) \wedge \forall k(i \leq k \leq n \rightarrow B(k) = A(k))\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i aldagaiaren balioari 1 kendu zaiola kontuan hartuz kalkulatuko da. Baina (5) puntuan i-ren balio berriak adierazten duen posizioko elementua oraindik ez da B-ra kopiatu. Beraz, i-ren balioari 1 kendu ondoren i-ren balioa n + 1 ez dela izango eta orain 1 ere izan litekeela baiezta dezakegu. Bestalde, A-tik B-ra kopiatutako posizioak i + 1 eta n-ren artekoak dira.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(i + 1 \leq k \leq n \rightarrow B(k) = A(k))\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat eskuinetik ezkerrera zeharkatzen ari garenean E espresioa "i ken i aldagaiak hartuko duen azkeneko balioa" izaten da. Beraz kasu honetan E = i – 1.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.11. B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa(eskuinetik ezkerrera zeharkatuz) (2. bertsioa)

Gutxienez elementu bat (n ≥ 1) duen A(1..n) bektorea emanda, B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen honako programa hau dokumentatu nahi da hor zehazten diren formulak emanez. Bektorea eskuinetik ezkerrera zeharkazten da eta programa hau 2.6.8, 2.6.9 eta 2.6.10 ataletan aurkeztutako programen antzekoa izan arren, ez da berdina eta desberdintasun horrek dokumentazioan duen eragina zein den ikusi nahi da. Lau programek emaitza bera kalkulatzen dute baina era desberdinean.

```
(1) ≡ {Hasierako baldintza}
i : = n;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i > 0 loop
        (4) ≡ {Tarteko asertzioa}
        B(i) : = A(i);
        (5) ≡ {Tarteko asertzioa}
        i : = i – 1;
end loop;
(6) ≡ {Bukaerako baldintza}
(7) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (4) eta (5) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

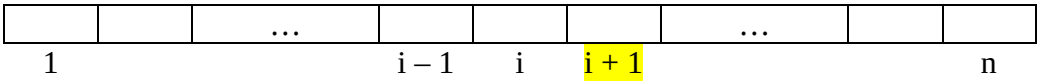
- (1) ≡ {n ≥ 1}
  (6) ≡ {∀k(1 ≤ k ≤ n → B(k) = A(k))}

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (6) puntuko formularen bidez, amaieran A(1..n) eta B(1..n) bektoreetan elementu berdinak edukiko ditugula adierazten da. Hasierako baldintza eta bukaerako baldintza 2.6.8, 2.6.9 eta 2.6.10 ataletan emandakoen berdinak dira, lau programak egoera beretik abiatzen direlako eta emaitza bera kalkulatzen dutelako, nahiz eta era desberdinean egin.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari n balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak n balio duela baiezta dezakegu:

  (2) ≡ {n ≥ 1 ∧ i = n}

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak n eta 0-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian n balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian 0 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioko elementua B-ko i posiziora kopiatzea denez, (3) puntuan gaudenean A(1..n) bektoreko i + 1 eta n posizioen arteko elementuak A-tik B-ra kopiatuta daude.

  $(3) \equiv \{(0 \leq i \leq n) \wedge \forall k(i + 1 \leq k \leq n \rightarrow B(k) = A(k))\}$

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa 0 baino handiagoa dela eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(i + 1 \leq k \leq n \rightarrow B(k) = A(k))\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i posizioko elementua B-ra kopiatu dela kontuan hartuz emango da. Baina (5) puntuan i-ri oraindik ez zaio 1 kendu.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(i \leq k \leq n \rightarrow B(k) = A(k))\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat eskuinetik ezkerrera zeharkatzen ari garenean E espresioa "i ken i aldagaiak hartuko duen azkeneko balioa" izaten da. Beraz kasu honetan E = i – 0, hau da, E = i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.12. B(1..n) bektorean A(1..n) bektoreko elementuak posizio bat ezkerrera biratuta gordetzen dituen programa

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea hartuta, A(1..n) bektoreko elementuak B(1..n) bektorean posizio bat ezkerrera biratuta gordetzen dituen honako programa dokumentatu nahi da adibide honetan.

```
(1) ≡ {Hasierako baldintza}
B(n) := A(1);
(2) ≡ {Tarteko asertzioa}
i : = 1;
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariantea} i < n loop
        (5) ≡ {Tarteko asertzioa}
        B(i) : = A(i + 1);
        (6) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. Bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (5) puntuko asertzioa inbariantetik abiatuta kalkulatuko da eta (6) puntuko asertzioa (5) puntuko asertziotik abiatuz kalkulatuko da. Bukatzeko E espresioa eman beharko da:

- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{B(n) = A(1) \wedge \forall k(1 \leq k \leq n - 1 \rightarrow B(k) = A(k + 1))\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela adierazten da eta bukaerako baldintzaren bidez, hau da, (7) puntuko formularen bidez, amaieran B(1..n) bektorean A(1..n) bektoreko elementuak ezkerrera posizio bat biratuta egongo direla adierazten da.

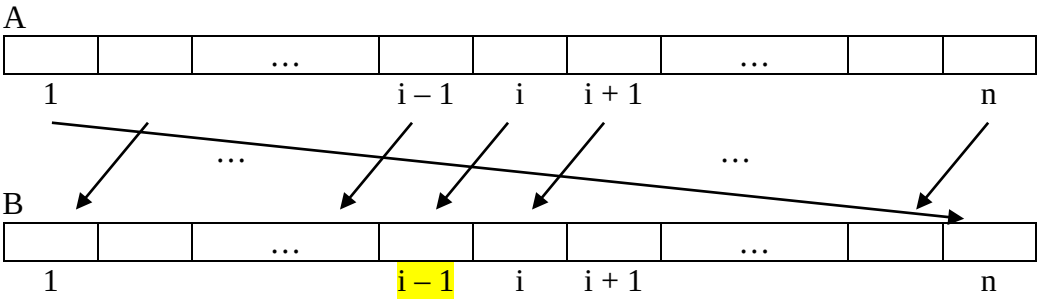
- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta B taulako n posizioan A(1) balioa gorde dela kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta B(n) eta A(n) berdinak direla baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge B(n) = A(1)\}$

- (3) puntuko asertzioa (2) puntuko asertziotik abiatuta eta i aldagaiari 1 balioa esleitu zaiola kontuan hartuz kalkulatuko da. (3) puntuan n 1 baino handiagoa edo berdina dela, B(n) eta A(1) berdinak direla eta i aldagaiak 1 balio duela baiezta dezakegu:

  $(3) \equiv \{n \geq 1 \wedge B(n) = A(1) \wedge i = 1\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 1 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza B-ko i posizioan A-ko i + 1 posizioko balioa gordetzea denez, (4) puntuan gaudenean B(1..n) bektoreko 1 eta i – 1 posizioetan A-ko 2 eta i posizioen arteko elementuak kopiatuta daude.

  $(4) \equiv \{B(n) = A(1) \wedge (1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k + 1))\}$

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n baino txikiagoa dela eta hori da hain zuzen ere (5) puntuaren eta (4) puntuaren arteko desberdintasuna.

  $(5) \equiv \{B(n) = A(1) \wedge (1 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow B(k) = A(k + 1))\}$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta A-ko i + 1 posizioko elementua B-ko i posiziora kopiatu dela kontuan hartuz kalkulatuko da.

  $(6) \equiv \{B(n) = A(1) \wedge (1 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i \rightarrow B(k) = A(k + 1))\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.13. A(1..n) bektoreko elementuak A(1..n) bektorean bertan ezkerrera biratzen dituen programa

Gutxienez elementu bat (n ≥ 1) duen A(1..n) bektorea emanda, A(1..n) bektoreko elementuak ezkerrera biratzen dituen honako programa hau dokumentatuko da jarraian.

```
(1) ≡ {Hasierako baldintza}
lag := A(1);
(2) ≡ {Tarteko asertzioa}
i : = 1;
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariantea} i < n loop
        (5) ≡ {Tarteko asertzioa}
        A(i) : = A(i + 1);
        (6) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(7) ≡ {Tarteko asertzioa}
A(n) := lag;
(8) ≡ {Bukaerako baldintza}
(9) ≡ {E espresioa}
```

Hasteko hasierako baldintza, bukaerako baldintza eta (7) puntuko asertzioa eman beharko dira, (7) puntuko asertzioa baita while-aren benetako "bukaerako baldintza". Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. (7) puntuko asertzioa kontuan hartuz inbariantea lortu beharko da. Gero, (5) puntuko asertzioa inbariantetik abiatuta kalkulatuko da eta (6) puntuko asertzioa (5) puntuko asertziotik abiatuz kalkulatuko da. Bukatzeko E espresioa eman beharko da:

- $(1) \equiv \{n \geq 1 \wedge \forall k(1 \leq k \leq n \rightarrow A(k) = a_k)\}$
  $(8) \equiv \{A(n) = a_1 \wedge \forall k(1 \leq k \leq n - 1 \rightarrow A(k) = a_{k+1})\}$
  $(7) \equiv \{lag = a_1 \wedge \forall k(1 \leq k \leq n - 1 \rightarrow A(k) = a_{k+1})\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela eta hasierako balioak a_1, …, a_n direla adierazten da. Bukaerako baldintzaren bidez, hau da, (8) puntuko formularen bidez, amaieran A(1..n) bektorean a_1, …, a_n hasierako balioak baina ezkerrera posizio bat biratuta egongo direla adierazten da. (7) puntuko asertzioak while-a bukatu ondoren a_2, …, a_n elementuek 1 eta (n

– 1)-en arteko posizioetan daudela eta a_1 balioa lag aldagaian dagoela adierazten du.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta lag aldagaiari A(1) balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta lag aldagaiak A(1) balio duela baiezta dezakegu:

  (2) ≡ {n ≥ 1 ∧ lag = A(1) = a_1}

- (3) puntuko asertzioa (2) puntuko asertziotik abiatuta eta i aldagaiari 1 balioa esleitu zaiola kontuan hartuz kalkulatuko da. (3) puntuan n 1 baino handiagoa edo berdina dela, lag eta A(1) eta a_1 berdinak direla eta i aldagaiak 1 balio duela baiezta dezakegu:

  (3) ≡ {n ≥ 1 ∧ lag = A(1) = a_1 ∧ i = 1}

- Inbariantea (7) puntuko asertzioa kontuan hartuz lortuko da. (7) puntuko asertzioan while-a bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.

  A

a_2	a_3	…	a_i	a_i	a_{i+1}	…		a_n
1			i – 1	i	i + 1			n

  Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 1 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioan A-ko i + 1 posizioko balioa gordetzea denez, (4) puntuan gaudenean A(1..n) bektoreko 1 eta i – 1 posizioetan hasieran A-ko 2 eta i posizioen artean zeuden elementuak kopiatuta daude.

  (4) ≡ {lag = a_1 ∧ (1 ≤ i ≤ n) ∧ ∀k(1 ≤ k ≤ i – 1 → A(k) = a_{k+1})}

  Orain ezin da esan lag eta A(1) berdinak direnik, A(1)-en balioa aldatuta egongo baita eta horregatik lag-en balioa hasieran A-ko 1 posizioan zegoen balioa da, hau da, a_1.

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n baino txikiagoa dela eta hori da hain zuzen ere (5) puntuaren eta (4) puntuaren arteko desberdintasuna.

  (5) ≡ {lag = a_1 ∧ (1 ≤ i ≤ n – 1) ∧ ∀k(1 ≤ k ≤ i – 1 → A(k) = a_{k+1})}

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta A-ko i + 1 posizioko elementua A-ko i posiziora kopiatu dela kontuan hartuz kalkulatuko da.

  $(6) \equiv \{lag = a_1 \wedge (1 \leq i \leq n - 1) \wedge \forall k(1 \leq k \leq i \rightarrow A(k) = a_{k+1})\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i eta lag aldagaiekin. Enuntziatuan ere ez dira agertzen i eta lag aldagaiak eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez direla agertuko. Baina inbariantean eta tarteko asertzioetan agertzen dira emaitza kalkulatzeko prozesuan zehar erabiltzen direlako eta garrantzitsuak direlako.

2.6.14. c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen programa (1. bertsioa)

Gutxienez elementu bat (n ≥ 1) duen A(1..n) bektorea emanda, c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen honako programa hau dokumentatuko da jarraian.

```
(1) ≡ {Hasierako baldintza}
i : = 1;
(2) ≡ {Tarteko asertzioa}
c : = 0;
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariantea} i <= n loop
        (5) ≡ {Tarteko asertzioa}
        if A(i) = 0 then (6) ≡ {Tarteko asertzioa}
                        c : = c + 1;
        end if;
        (7) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(8) ≡ {Bukaerako baldintza}
(9) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. Bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (5) puntuko asertzioa inbariantetik abiatuta kalkulatuko da eta (6) puntuko asertzioa (5) puntuko asertziotik abiatuz kalkulatuko da. (7) puntuko asertzioa ere (5) puntuko asertziotik abiatuz kalkulatuko da. Bukatzeko E espresioa eman beharko da:

- (1) ≡ {n ≥ 1}
  (8) ≡ {c = Nk(1 ≤ k ≤ n ∧ A(k) = 0)}

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela adierazten da. Bukaerako baldintzaren bidez, hau da, (8) puntuko formularen bidez, amaieran c aldagaian A(1..n) bektoreko zero-kopurua edukiko dela adierazten da.

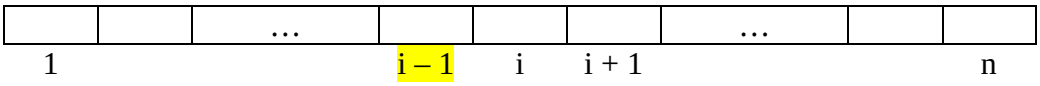
- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 1 balio duela baiezta dezakegu:

  (2) ≡ {n ≥ 1 ∧ i = 1}

- (3) puntuko asertzioa (2) puntuko asertziotik abiatuta eta c aldagaiari 0 balioa esleitu zaiola kontuan hartuz kalkulatuko da:

  (3) ≡ {n ≥ 1 ∧ i = 1 ∧ c = 0}

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioko balioa 0rekin konparatzea denez, (4) puntuan gaudenean A(1..n) bektoreko 1 eta i – 1 posizioen arteko elementuak 0rekin konparatuta daude.

  (4) ≡ {(1 ≤ i ≤ n + 1) ∧ c = Nk(1 ≤ k ≤ i – 1 ∧ A(k) = 0)}

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n + 1 baino txikiagoa dela eta hori da hain zuzen ere (4) puntuaren eta (4) puntuaren arteko desberdintasuna.

  (5) ≡ {(1 ≤ i ≤ n) ∧ c = Nk(1 ≤ k ≤ i – 1 ∧ A(k) = 0)}

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i posizioko elementua 0 dela baina c oraindik ez dela eguneratu kontuan hartuz kalkulatuko da. Beraz 1

eta i posizioen arteko elementuak aztertu dira eta gainera badakigu i posiziokoa 0 dela, baina c ez da eguneratu oraindik.

$(6) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i - 1 \wedge A(k) = 0) \wedge A(i) = 0\}$

A(k) 0 denez eta c aldagaian 1 eta i – 1 posizioen arteko zeroak ditugunez zenbatuta, 1 eta i posizioen arteko zero-kopurua c + 1 dela esan dezakegu. Beraz, (6) puntuko asertzioa honela ere idatz daiteke:

$(6) \equiv \{(1 \leq i \leq n) \wedge c + 1 = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- (7) puntuko asertzioa (4) puntuko asertziotik abiatuz eta if agindua burutu dela kontuan hartuz kalkulatuko da. Puntu honetan ez dakigu i posizioko elementua 0 al den ala ez baina badakigu c eguneratua izan dela.

  $(7) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n + 1 – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.15. c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen programa (2. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, c aldagaian A(1..n) bektoreko zero kopurua kalkulatzen duen honako programa hau dokumentatuko da orain. Programa honek 4.2.9 ataleko programak lortzen duen emaitza bera kalkulatzen du baina era desberdinean. Desberdintasun horrek dokumentazioan duen eragina ondo ulertzea komeni da.

```
(1) ≡ {Hasierako baldintza}
i : = 0;
(2) ≡ {Tarteko asertzioa}
c : = 0;
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariantea} i < n loop
        (5) ≡ {Tarteko asertzioa}
        i : = i + 1;
        (6) ≡ {Tarteko asertzioa}
        if A(i) = 0 then (7) ≡ {Tarteko asertzioa}
                        c : = c + 1;
        end if;
end loop;
(8) ≡ {Bukaerako baldintza}
(9) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. Bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (5) puntuko asertzioa inbariantetik abiatuta kalkulatuko da eta (6) puntuko asertzioa (5) puntuko asertziotik abiatuz kalkulatuko da. (7) puntuko asertzioa (6) puntuko asertziotik abiatuz kalkulatuko da. Bukatzeko E espresioa eman beharko da:

- $(1) \equiv \{n \geq 1\}$
  $(8) \equiv \{c = Nk(1 \leq k \leq n \wedge A(k) = 0)\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrera bezala gutxienez elementu bat duen bektore bat izango duela adierazten da. Bukaerako baldintzaren bidez, hau da, (8) puntuko formularen bidez, amaieran c aldagaian A(1..n) bektoreko zero-kopurua edukiko dela adierazten da. Hasierako eta bukaerako baldintza aurreko ataleko adibidean emandakoen berdinak dira programa biak egoera beretik abiatzen direlako eta emaitza bera kalkulatzen dutelako.

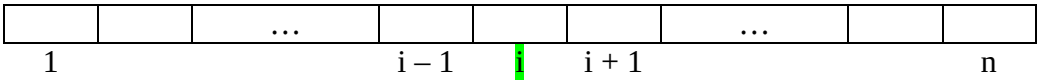
- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 0 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 0 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 0\}$

- (3) puntuko asertzioa (2) puntuko asertziotik abiatuta eta c aldagaiari 0 balioa esleitu zaiola kontuan hartuz kalkulatuko da:

  $(3) \equiv \{n \geq 1 \wedge i = 0 \wedge c = 0\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 0 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 0 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza i-ri 1 gehitzea denez, (4) puntuan gaudenean 1 eta i posizioen arteko elementuak dagoeneko orekin konparatu dira.

  $(4) \equiv \{(0 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n baino txikiagoa dela eta hori da hain zuzen ere (5) puntuaren eta (4) puntuaren arteko desberdintasuna.

  $(5) \equiv \{(0 \leq i \leq n - 1) \wedge c = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i aldagaiaren balioari 1 gehitu zaiola kontuan hartuz kalkulatuko da. Baina (6) puntuan i-ren balio berriak adierazten duen posizioko elementua oraindik ez da aztertu. Beraz, i-ren balioari 1 gehitu ondoren i-ren balioa 0 ez dela izango eta orain n ere izan litekeela baiezta dezakegu eta c aldagaian 1 eta i – 1 posizioen arteko zero-kopurua edukiko dugu:

  $(6) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i - 1 \wedge A(k) = 0)\}$

- (7) puntuko asertzioa (6) puntuko asertziotik abiatuz eta A(i)-ren balioa 0 dela baina c oraindik ez dela eguneratu kontsideratuz kalkulatuko da. Beraz 1 eta i-ren arteko posizioak aztertu dira eta i posizioan 0 balioa dago.

  $(7) \equiv \{(1 \leq i \leq n) \wedge c = Nk(1 \leq k \leq i - 1 \wedge A(k) = 0) \wedge A(i) = 0\}$

  A(k) 0 denez eta c aldagaian 1 eta i – 1 posizioen arteko zeroak ditugunez zenbatuta, 1 eta i posizioen arteko zero kopurua c + 1 dela esan dezakegu. Beraz, (6) puntuko asertzioa honela ere idatz daiteke:

$(7) \equiv \{(1 \leq i \leq n) \wedge c + 1 = Nk(1 \leq k \leq i \wedge A(k) = 0)\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.16. b aldagaian A(1..n) bektorean zerorik agertzen al den ala ez erabakitzen duen programa (1. bertsioa)

Gutxienez elementu bat ($n \geq 1$) duen A(1..n) bektorea emanda, b aldagai boolearrean A(1..n) bektorean zerorik ba al dagoen ala ez erabakitzen duen honako programa hau dokumentatu nahi da:

```
(1) ≡ {Hasierako baldintza}
i : = 1;
(2) ≡ {Tarteko asertzioa}
b : = false;
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariantea} i <= n and not b loop
        (5) ≡ {Tarteko asertzioa}
        b : = (A(i) = 0);
        (6) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. Bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (5) puntuko asertzioa inbariantetik abiatuta kalkulatuko da eta (6) puntuko asertzioa (5) puntuko asertziotik abiatuz kalkulatuko da. Bukatzeko E espresioa eman beharko da:

- $(1) \equiv \{n \geq 1\}$
  $(7) \equiv \{b \leftrightarrow \exists k(1 \leq k \leq n \wedge A(k) = 0)\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako gisa gutxienez elementu bat duen bektore bat izango duela adierazten

da. Bukaerako baldintzaren bidez, hau da, (7) puntuko formularen bidez, A(1..n) bektorean zerorik badago amaieran b aldagaiak True balioko duela eta bestela False balioko duela adierazten da.

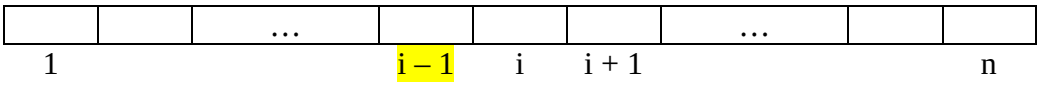
- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 1 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- (3) puntuko asertzioa (2) puntuko asertziotik abiatuta eta b aldagaiari False balioa esleitu zaiola kontuan hartuz kalkulatuko da. (3) puntuan n 1 baino handiagoa edo berdina dela, i aldagaiak 1 balio duela eta b aldagaiak False balio duela baiezta dezakegu:

  $(3) \equiv \{n \geq 1 \wedge i = 1 \wedge \neg b\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioko balioa 0rekin konparatzea denez, (4) puntuan gaudenean A(1..n) bektoreko 1 eta i – 1 posizioen arteko elementuak konparatuta daude baina i posiziokoa ez.

  $(4) \equiv \{(1 \leq i \leq n + 1) \wedge b \leftrightarrow \exists k(1 \leq k \leq i - 1 \wedge A(k) = 0)\}$

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n baino txikiagoa edo berdina dela eta hori da hain zuzen ere (5) puntuaren eta (4) puntuaren arteko desberdintasuna.

  $(5) \equiv \{(1 \leq i \leq n) \wedge b \leftrightarrow \exists k(1 \leq k \leq i - 1 \wedge A(k) = 0)\}$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i posizioko elementua 0 al den begiratu ondoren b aldagaia eguneratua izan dela kontuan hartuz kalkulatuko da.

  $(6) \equiv \{(1 \leq i \leq n) \wedge b \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = 0)\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n + 1 – i. Bektore osoa zeharkatu baino lehen aurkitu dezakegunez 0 balioa, eta hori gertatzen bada while-a bektore osoa zeharkatu baino lehenago bukatuko denez, E espresioak gehienez zenbat buelta gelditzen diren adierazten digu.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.17. b aldagaian A(1..n) bektorean zerorik agertzen al den ala ez erabakitzen duen programa (2. bertsioa)

Gutxienez elementu bat (n ≥ 1) duen A(1..n) bektorea emanda, b aldagai boolearrean A(1..n) bektorean zerorik ba al dagoen ala ez erabakitzen duen honako programa hau dokumentatuko da orain. Aurre-ondoetako espezifikazioa 4.2.9 ataleko programakoaren berdina izango da biak egoera beretik abiatzen baitira eta emaitza bera kalkulatzen baitute. Baina kalkuluak era desberdinean egiten dituztenez, inbariantea eta tarteko asertzioak desberdinak izango dira:

```
(1) ≡ {Hasierako baldintza}
i : = 1;
(2) ≡ {Tarteko asertzioa}
b : = (A(1) = 0);
(3) ≡ {Tarteko asertzioa}
while (4) ≡ {Inbariantea} i ≠ n and not b loop
        (5) ≡ {Tarteko asertzioa}
        b : = (A(i + 1) = 0);
        (6) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(7) ≡ {Bukaerako baldintza}
(8) ≡ {E espresioa}
```

Hasteko hasierako baldintza eta bukaerako baldintza eman beharko dira. Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortuko da, (2) puntuko asertzioa kontuan hartuz (3) puntuko asertzioa lortuko da. Bukaerako baldintza kontuan hartuz inbariantea lortu beharko da. Gero, (5) puntuko asertzioa inbariantetik abiatuta kalkulatuko da eta (6) puntuko asertzioa (5) puntuko asertziotik abiatuz kalkulatuko da. Bukatzeko E espresioa eman beharko da:

- (1) ≡ {n ≥ 1}

$(7) \equiv \{b \leftrightarrow \exists k(1 \leq k \leq n \wedge A(k) = 0)\}$

(1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela adierazten da. Bukaerako baldintzaren bidez, hau da, (7) puntuko formularen bidez, A(1..n) bektorean zerorik badago amaieran b aldagaiak True balioko duela eta bestela False balioko duela adierazten da.

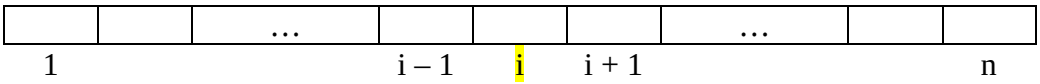
- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa esleitu zaiola kontuan hartu beharko da. (2) puntuan n 1 baino handiagoa edo berdina dela eta i aldagaiak 1 balio duela baiezta dezakegu:

  $(2) \equiv \{n \geq 1 \wedge i = 1\}$

- (3) puntuko asertzioa (2) puntuko asertziotik abiatuta eta b aldagaiari A(1) eta 0 konparatzearen emaitza esleitu zaiola kontuan hartuz kalkulatuko da. (3) puntuan n 1 baino handiagoa edo berdina dela, i aldagaiak 1 balio duela eta b aldagaiak A(1= eta 0 berdinak badira True eta bestela False balio duela esan dezakegu:

  $(3) \equiv \{n \geq 1 \wedge i = 1 \wedge b \leftrightarrow (A(1) = 0)\}$

- Inbariantea bukaerako baldintza kontuan hartuz lortuko da. Bukaerako baldintzan programa bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



Beraz inbariantean, hau da, programako (4) puntuan i aldagaiak 1 eta n-ren arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i + 1 posizioko balioa 0rekin konparatzea denez, (4) puntuan gaudenean A(1..n) bektoreko 1 eta i posizioen arteko elementuak konparatuta daude baita i posiziokoa ere.

$(4) \equiv \{(1 \leq i \leq n) \wedge b \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = 0)\}$

- (5) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (5) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n + 1 baino txikiagoa dela eta hori da hain zuzen ere (5) puntuaren eta (4) puntuaren arteko desberdintasuna.

  $(5) \equiv \{(1 \leq i \leq n - 1) \wedge b \leftrightarrow \exists k(1 \leq k \leq i \wedge A(k) = 0)\}$

- (6) puntuko asertzioa (5) puntuko asertziotik abiatuz eta i + 1 posizioko elementua 0 al den begiratu ondoren b aldagaia eguneratua izan dela kontuan hartuz kalkulatuko da.

  $(6) \equiv \{((1 \leq i \leq n - 1) \wedge b \leftrightarrow \exists k(1 \leq k \leq i + 1 \wedge A(k) = 0)\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (4) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n – i. Bektore osoa zeharkatu baino lehen aurkitu dezakegunez 0 balioa, eta hori gertatzen bada while-a bektore osoa zeharkatu baino lehenago bukatuko denez, E espresioak gehienez zenbat buelta gelditzen diren adierazten digu.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i aldagaiarekin. Enuntziatuan ere ez da agertzen i aldagaia eta horrek esan nahi du hasierako eta bukaerako baldintzetan ere ez dela agertuko. Baina inbariantean eta tarteko asertzioetan agertzen da emaitza kalkulatzeko prozesuan zehar erabiltzen delako eta garrantzitsua delako.

2.6.18. b aldagaian A(1..n) bektoreko bikoiti kopurua bakoiti kopurua baino handiagoa al den erabakitzen duen programa

Sarrera bezala A(1..n) bektore bat emanda eta n ≥ 1 dela jakinda, b aldagai boolearrean A(1..n) bektorean elementu bikoitien kopurua bakoitien kopurua baino handiagoa al den erabakitzen duen honako programa honi dagokion dokumentazioa emango da jarraian:

```
(1) ≡ {Hasierako baldintza}
i : = 1;
bik: = 0;
bak := 0;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} i <= n loop
        (4) ≡ {Tarteko asertzioa}
        if A(i) mod 2 = 0 then (5) ≡ {Tarteko asertzioa}
                                bik : = bik + 1;
        else (6) ≡ {Tarteko asertzioa}
                bak : = bak + 1;
        end if;
        (7) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(8) ≡ {Tarteko asertzioa}
b : = (bik > bak);
(9) ≡ {Bukaerako baldintza}
(10) ≡ {E espresioa}
```

Hasteko hasierako baldintza, bukaerako baldintza eta (8) puntuko asertzioa eman beharko dira, (8) puntuko asertzioa baita while-aren benetako "bukaerako baldintza". Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta (8) puntuko asertzioa kontuan hartuz inbariantea lortu beharko da. Gero, (4) eta (5) asertzioak inbariantetik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko da:

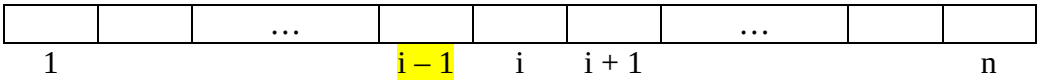
- (1) ≡ {n ≥ 1}
  (9) ≡ {b ↔ Nk(1 ≤ k ≤ n ∧ bikoitia(A(k))) > Nk(1 ≤ k ≤ n ∧ ¬bikoitia(A(k)))}
  (8) ≡ {bik = Nk(1 ≤ k ≤ n ∧ bikoitia(A(k))) ∧ bak = Nk(1 ≤ k ≤ n ∧ ¬bikoitia(A(k)))}

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela adierazten da. Bukaerako baldintzaren bidez, hau da, (9) puntuko formularen bidez, A(1..n) bektorean elementu bikoitien kopurua bakoitiena baino handiagoa bada, amaieran b aldagaiak True balioko duela eta bestela False balioko duela adierazten da. (8) puntuko formulak while-a bukatu ondoren bik-en bikoiti kopurua eta bak-en bakoiti kopurua edukiko direla adierazten du.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa, bik aldagaiari 0 balioa eta bak aldagaiari 0 balioa esleitu zaiela kontuan hartu beharko da:

(2) ≡ {n ≥ 1 ∧ i = 1 ∧ bik = 0 ∧ bak = 0}

- Inbariantea (8) puntuko asertzioa kontuan hartuz lortuko da. (8) puntuko asertzioan while-a bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioko balioa bikoitia al den ala ez erabakitzea denez, (3) puntuan gaudenean A(1..n) bektoreko 1 eta i – 1 posizioen arteko elementuak aztertuta daude baina i posiziokoa ez.

(3) ≡ {(1 ≤ i ≤ n + 1) ∧ bik = Nk(1 ≤ k ≤ i – 1 ∧ bikoitia(A(k))) ∧ bak = Nk(1 ≤ k ≤ i – 1 ∧ ¬bikoitia(A(k)))}

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n + 1 baino txikiagoa dela eta hori da hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasuna.

(4) ≡ {(1 ≤ i ≤ n) ∧ bik = Nk(1 ≤ k ≤ i – 1 ∧ bikoitia(A(k))) ∧ bak = Nk(1 ≤ k ≤ i – 1 ∧ ¬bikoitia(A(k)))}

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i posizioko elementua bikoitia dela baina bik oraindik ez dela eguneratu kontuan hartuz kalkulatuko da.

(5) ≡ {(1 ≤ i ≤ n) ∧ bikoitia(A(i)) ∧ bik = Nk(1 ≤ k ≤ i – 1 ∧ bikoitia(A(k))) ∧ bak = Nk(1 ≤ k ≤ i – 1 ∧ ¬bikoitia(A(k))) }

Baina bik aldagaian 1 eta i – 1 posizioen arteko bikoiti-kopurua daukagula, bak aldagaian 1 eta i – 1 posizioen arteko bakoiti kopurua daukagula eta i posiziokoa bikoitia dela jakinda, honako formula hau ere egokia da (5) punturako:

(5) ≡ {(1 ≤ i ≤ n) ∧ bik + 1 = Nk(1 ≤ k ≤ i ∧ bikoitia(A(k))) ∧ bak = Nk(1 ≤ k ≤ i ∧ ¬bikoitia(A(k)))}

- (6) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i posizioko elementua bakoitia dela baina bak oraindik ez dela eguneratu kontuan hartuz kalkulatuko da.

(6) ≡ {(1 ≤ i ≤ n) ∧ ¬bikoitia(A(i)) ∧ bik = Nk(1 ≤ k ≤ i – 1 ∧ bikoitia(A(k))) ∧ bak = Nk(1 ≤ k ≤ i – 1 ∧ ¬bikoitia(A(k))) }

Baina bik aldagaian 1 eta i – 1 posizioen arteko bikoiti kopurua daukagula, bak aldagaian 1 eta i – 1 posizioen arteko bakoiti kopurua daukagula eta i posiziokoa bakoitia dela jakinda, honako formula hau ere egokia da (6) punturako:

(6) ≡ {(1 ≤ i ≤ n) ∧ bik = Nk(1 ≤ k ≤ i ∧ bikoitia(A(k))) ∧ bak + 1 = Nk(1 ≤ k ≤ i ∧ ¬bikoitia(A(k)))}

- (7) puntuko asertzioa (4) puntuko asertziotik abiatuz kalkulatuko da. (7) puntuan ez dakigu i posizioko elementua bikoitia ala bakoitia den baina badakigu bik eta bak aldagaiak eguneratuta daudela. Beraz 1 eta i posizioen arteko elementuak aztertuta daude eta bik eta bak eguneratuta.

  (7) ≡ {(1 ≤ i ≤ n) ∧ bik = Nk(1 ≤ k ≤ i ∧ bikoitia(A(k))) ∧ bak = Nk(1 ≤ k ≤ i ∧ ¬bikoitia(A(k)))}

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n + 1 – i. Programa honekin beti bektore osoa zeharkatuko denez, justu zenbat buelta gelditzen diren adieraziko digu E espresioak adibide honetan.

Aurre-ondoetako espezifikazioak sarrerako datuak eta bukaerako emaitza deskribatzeko balio duenez, hasierako baldintzan eta bukaerako baldintzan ez dira agertuko programaren barruan laguntzaile bezala erabili diren aldagaiak. Kasu honetan hori gertatzen da i, bik eta bak aldagaiekin.

Sarrerako datua A(1..n) da eta emaitza b da. Beste hiru aldagaiak ez dira aurre-ondoetako espezifikazioan agertzen, izan ere aurre-ondoetako espezifikazioak sarrerako eta irteerako datuak deskribatzeko balio du.

Bestalde b aldagaia bukaeako baldintzara arte ez da agertzen while-aren barruan ez baita erabili.

2.6.19. x balioa A(1..n) bektorean agertzen dela jakinda, bere lehenengo agerpenaren posizioa pos aldagaian kalkulatzen duen programa

x balioa gutxienez elementu bat duen A(1..n) bektorea emanda eta x balioa A(1..n) bektorean gutxienez behin agertzen dela jakinda, x-en lehenengo agerpenaren posizioa pos aldagaian kalkulatzen duen honako programa hau dokumentatuko da jarraian.

```
(1) ≡ {Hasierako baldintza}
i : = 1;
segi: = true;
(2) ≡ {Tarteko asertzioa}
while (3) ≡ {Inbariantea} segi loop
        (4) ≡ {Tarteko asertzioa}
        if A(i) = x then (5) ≡ {Tarteko asertzioa}
                                segi : = false;
        end if;
        (6) ≡ {Tarteko asertzioa}
        i : = i + 1;
end loop;
(7) ≡ {Tarteko asertzioa}
pos : = i – 1;
(8) ≡ {Bukaerako baldintza}
(9) ≡ {E espresioa}
```

Hasteko hasierako baldintza, bukaerako baldintza eta (7) puntuko asertzioa eman beharko dira, (7) puntuko asertzioa baita while-aren benetako "bukaerako baldintza". Hasierako baldintza kontuan hartuz (2) puntuko asertzioa lortu beharko da eta (7) puntuko asertzioa kontuan hartuz inbariantea lortu beharko da. Gero, (4) puntuko asertzioa inbariantetik abiatuz lortuko da eta (5) eta (6) puntuetako asertzioak (4) puntuko asertziotik abiatuz kalkulatuko dira. Bukatzeko E espresioa eman beharko:

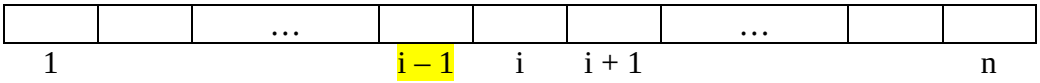
- $(1) \equiv \{n \geq 1 \wedge \exists k(1 \leq k \leq n \wedge A(k) = x)\}$
  $(8) \equiv \{1 \leq pos \leq n \wedge A(pos) = x \wedge \forall k(1 \leq k \leq pos - 1 \rightarrow A(k) \neq x)\}$
  $(7) \equiv \{1 \leq i \leq n + 1 \wedge A(i - 1) = x \wedge \forall k(1 \leq k \leq i - 2 \rightarrow A(k) \neq x) \wedge \neg segi\}$

  (1) puntuko formularen bidez, hau da, hasierako baldintzaren bidez, programak sarrerako datu gisa gutxienez elementu bat duen bektore bat izango duela eta bektore horretan x gutxienez behin agertuko dela adierazten da. Bukaerako baldintzaren bidez, hau da, (8) puntuko formularen bidez, bukaeran pos aldagaian x elementuaren lehenengo agerpenari dagokion posizioa edukiko dela adierazten da. (7) puntuko formularen bidez, while-aren bukaeran i – 1 posizioan x-en lehenengo agerpena daukagula adierazten da.

- (2) puntuko asertzioa kalkulatzeko (1) puntuko hasierako baldintzatik abiatu eta i aldagaiari 1 balioa eta segi aldagaiari True balioa esleitu zaiola kontuan hartu beharko da:

  $(2) \equiv \{n \geq 1 \wedge i = 1 \wedge segi\}$

- Inbariantea (7) puntuko asertzioa kontuan hartuz lortuko da. (7) puntuko asertzioan while-a bukatu ondoren egoera zein den deskribatzen da eta inbariantean while-a exekutatzen ari denean edozein bueltatan egoera zein den deskribatuko da. Beraz i aldagaia bektoreko erdialdeko posizio bat apuntatzen duela kontsideratu beharko da eta egoera hori deskribatuko da.



  Beraz inbariantean, hau da, programako (3) puntuan i aldagaiak 1 eta n + 1-en arteko balio bat izango duela esan dezakegu: while-aren baldintza aztertzen den lehenengo aldian 1 balioa izango du eta while-aren baldintza aztertzen den azkeneko aldian gehienez n + 1 balioa izango du. Bestalde, while-ean sartu ondoren egiten den lehenengo gauza A-ko i posizioko balioa x-ekin konparatzea denez, (3) puntuan gaudenean A(1..n) bektoreko 1 eta i – 1 posizioen arteko elementuak konparatuta daude baina i posiziokoa ez. While-an sartzeko "segi" aldagaiak True balio beharko du. "segi" aldagaiak False balio badu x-en lehenengo agerpena i – 1 posizioan egongo da eta ondorioz badakigu [1..i – 2] tartean x ez dela agertu. "segi" aldagaiak True balio badu [1..i – 1] tartean ez da x agertu:

  $(3) \equiv \{(1 \leq i \leq n + 1) \wedge \forall k(1 \leq k \leq i - 2 \rightarrow A(k) \neq x) \wedge$
  $\wedge (segi \leftrightarrow A(i - 1) \neq x)\}$

  segi aldagaiak True balio badu, ez dakigu A(i) balioa x-en berdina al den ala ez oraindik ez baita i posizioa aztertu.

- (4) puntuko asertzioa inbariantetik abiatuz kalkulatuko da. (4) puntuan gaudenean badakigu while-aren baldintza bete egin dela eta ondorioz badakigu i-ren balioa n + 1 baino txikiagoa dela eta "segi" aldagaiaren balioa True dela. Horiek dira hain zuzen ere (4) puntuaren eta (3) puntuaren arteko desberdintasunak.

  $(4) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow A(k) \neq x) \wedge segi\}$

- (5) puntuko asertzioa (4) puntuko asertziotik abiatuz eta i posizioko elementua x dela baina segi aldagaia oraindik ez dela eguneratu kontuan hartuz kalkulatuko da.

  $(5) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow A(k) \neq x) \wedge segi \wedge A(i) = x\}$

- (6) puntuko asertzioa (4) puntuko asertziotik abiatuz kalkulatuko da, izan ere (6) puntuan ez dakigu ez dakigu i posizioan x al dagoen ala ez. Baina badakigu "segi" aldagaia eguneratuta dagoela. Beraz konparatutako elementuak 1 eta i-ren artekoak dira.

  $(6) \equiv \{(1 \leq i \leq n) \wedge \forall k(1 \leq k \leq i - 1 \rightarrow A(k) \neq x) \wedge (segi \leftrightarrow A(i) \neq x)\}$

- Amaitzeko E espresioa eman behar da. Inbariantea betetzen den puntuan, hau da, (3) puntuan gauden bakoitzean, E espresioak gehienez zenbat buelta gelditzen diren adierazi behar du. Bektore bat ezkerretik eskuinera zeharkatzen ari garenean E espresioa "i aldagaiak hartuko duen azkeneko balioa ken i" izaten da. Beraz kasu honetan E = n + 1 – i. Bektore osoa zeharkatu baino lehen buka daitekeenez while-a, gehienez zenbat buelta gelditzen diren adierazten digu E espresioak.

Adibide honetan segi eta i aldagaiak bukaerako emaitza kalkulatzeko prozesuan garrantzitsuak dira baina segi eta i emaitzak ez direnez bukaerako baldintzan ez dira agertzen. Bestalde pos aldagaia bukaerako baldintzan bakarrik agertzen da while-aren barruan ez baita erabili.

2.6.20. (2006 #1 - Partziala)

a) C(1..r) bektoreko elementu denak positiboak (> 0) direla adierazten duen **positiboak(C(1..r))** izeneko predikatua definitu.

b) C(1..r) bektorean z elementua agertzen den posizioetan D(1..r) bektoreak 0 balioa duela eta beste posizioetan bektore bietan elementu berdinak daudela adierazten duen **anulatuta(z, C(1..r), D(1..r))** izeneko predikatua definitu.

1. adibidea
z balioa 8 dela kontsideratuz, adibide honetako C(1..6) eta D(1..6) bektoreentzat predikatua bete egiten da:

C(1..6)	2	8	1	15	20	8
	1	2	3	4	5	6
D(1..6)	2	0	1	15	20	0
	1	2	3	4	5	6

2. adibidea
z balioa 15 dela kontsideratuz, adibide honetako C(1..6) eta D(1..6) bektoreentzat predikatua ez da betetzen:

C(1..6)	2	8	1	15	20	8
	1	2	3	4	5	6
D(1..6)	2	8	6	15	20	0
	1	2	3	4	5	6

c) C(1..r) bektorean z elementua v aldiz agertzen dela adierazten duen **aldiz(C(1..r), z, v)** izeneko predikatua definitu.

d) x zenbaki osoa eta 0 baino handiagoak diren balioz osatutako A(1..n) bektorea emanda, p aldagaian x balioa A(1..n) bektorean zenbat aldiz agertzen den kalkulatu eta x balioaren ordez 0 balioa ipiniz B(1..n) bektorean A(1..n) bektorearen kopia bat sortzen duen programa dokumentatu zehaztutako puntuetako asertzioak idatziz. Asertzioak idazterakoan aurreko ataletako predikatuak erabili behar dira.

```
(1) {Hasierako baldintza}
i : = 1;
p : = 0;
while (2) {Inbariantea} i ≠ n + 1 loop
    (3) {Tarteko asertzioa}
    if A(i) = x then (4) {Tarteko asertzioa}
                     B(i) : = 0;
                     (5) {Tarteko asertzioa}
                     p : = p + 1;
    else (6) {Tarteko asertzioa}
         B(i) : = A(i);
    end if;
    (7) {Tarteko asertzioa}
    i : = i + 1;
end loop;
(8) {Bukaerako baldintza}
(9) {E espresioa}
```

Soluzioa:

a) **positiboak(C(1..r))** ≡ {∀i (1 ≤ i ≤ r → C(i) > 0)}

b) **anulatuta(z, C(1..r), D(1..r))** ≡ {∀i ((1 ≤ i ≤ r ∧ C(i) = z) → D(i) = 0) ∧
∧ ∀i ((1 ≤ i ≤ r ∧ C(i) ≠ z) → D(i) = C(i))}

c) **aldiz(C(1..r), z, v)** ≡ {v = Ni (1 ≤ i ≤ r ∧ C(i) = z)}

d) Lehenengo hasierako baldintza, bukaerako baldintza eta inbariantea emango dira:

(1) {n ≥ 1 ∧ positiboak(A(1..n))}
(8) {aldiz(A(1..n), x, p) ∧ anulatuta(x, A(1..n), B(1..n))}
(2) {1 ≤ i ≤ n + 1 ∧ aldiz(A(1..i – 1), x, p) ∧
anulatuta(x, A(1..i – 1), B(1..i – 1))}

Jarraian while-aren barruan dauden asertzioak emango dira. Horretarako inbariantetik abiatuko gara eta puntu bakoitzean inbariantearekiko zer aldatu den zehaztu beharo da:

(3) {1 ≤ i ≤ **n** ∧ aldiz(A(1..i – 1), x, p) ∧ anulatuta(x, A(1.. i – 1), B(1..i – 1))}

(3) zenbakia dagoen lekuan badakigu while-an sartu garela eta beraz while-ko baldintza bete egin dela. Hori dela eta, orain badakigu i ≤ n dela eta hori izango da (3) puntuan (2) puntuarekiko aldatzen dena.

(4) {1 ≤ i ≤ n ∧ aldiz(A(1..i – 1), x, p) ∧ anulatuta(x, A(1..i – 1), B(1..i – 1)) ∧
A(i) = x}

(4) zenbakia dagoen lekuan gaudenean badakigu if aginduko then kasuan sartu garela eta beraz if-eko baldintza bete egin dela. Ondorioz puntu horretan A(i) = x beteko da eta hori da (4) puntuan (3) puntuarekiko aldatzen dena. Kasu honetan gauza bera adierazteko beste era bat ere badago:

{1 ≤ i ≤ n ∧ aldiz(A(1..**i**), x, **p + 1**) ∧ anulatuta(x, A(1..i – 1), B(1..i – 1))}

Bigarren bertsio hau lehenengoa baino hobea da.

(5) {1 ≤ i ≤ n ∧ aldiz(A(1..i), x, p + 1) ∧ anulatuta(x, A(1..**i**), B(1..**i**))}

(5) zenbakia dagoen lekuan gaudenean badakigu if aginduko then kasuan sartu garela eta gainera B taulako i posizioa eguneratu egin dela . (5) puntuko asertzioa (4) puntuko asertzioaren bigarren bertsioa eguneratuz lortu da.

(6) {1 ≤ i ≤ n ∧ aldiz(A(1..i), x, p) ∧ anulatuta(x, A(1..i – 1), B(1..i – 1))}

(6) zenbakia dagoen lekuan gaudenean badakigu if aginduaren else kasuan sartu garela baina oraindik B taulako i posizioa ez dela eguneratu. (6) puntuko asertzioa (3) puntuko asertzioa aldatuz lortu da (3) puntukoa baita bere aurreko asertzioa, izan ere (3) puntuan gaudenean if aginduko baldintzaren arabera (4) puntura edo (6) puntura goaz. Kasu honetan x balioa A(i)-ren desberdina denez p ez da eguneratu behar, edo hobeto esanda, p dagoeneko eguneratuta dago eta x balioa i posiziora arte (i bera ere barne) zenbat aldiz agertzen den adierazten du, eta hori da (3) puntuko asertzioarekiko aldatu dena.

(7) $\{1 \leq i \leq n \wedge \text{aldiz}(A(1..i), x, p) \wedge \text{anulatuta}(x, A(1..i), B(1..i))\}$

(7) puntuan ez dakigu if-aren then kasutik edo else kasutik joan garen baina badakigu B(i) eta p eguneratuta daudela eta A taula i posizioraino (i posizioa barne) aztertuta dagoela.

Bukatzeko E espresioa emango da:

(9) $\{E = n + 1 – i\}$
Inbariantea betetzen den puntuan gaudenean E espresioak zenbat buelta falta diren adieraziko digu era zehatzean.

2.6.21. (2006 #2 - Partziala)

a) x bikoitia dela adierazten duen **bikoitia(x)** izeneko predikatua definitu.

b) C(1..r) taulan 1 eta x – 1 posizioen artean (biak barne) dauden balio denak bikoitiak direla eta z eta r-ren artean daudenak (biak barne) bakoitiak direla adierazten duen **partizioa(C(1..r), x, z)** izeneko predikatua definitu. Aurreko ataleko predikatua erabili behar da.

c) C(1..r) bektorea (c_1, c_2, …, c_r) bektorearen permutazio bat dela adierazten duen **perm(C(1..r), (c_1, c_2, …, c_r))** izeneko predikatua definitu. C(1..r) bektorea (c_1, c_2, …, c_r) bektorearen permutazioa izateak C-ko elementu bakoitza C-n eta (c_1, c_2, …, c_r) bektorean kopuru berean agertzen dela esan nahi du.

1. adibidea
Adibide honetako C(1..6) bektorea (2, 8, 1, 15, 20, 8) bektorearen permutazioa da:

C(1..6)	2	20	1	8	8	15
	1	2	3	4	5	6

2. adibidea
Adibide honetako C(1..6) bektorea ez da (2, 8, 1, 15, 20, 8) bektorearen permutazioa:

C(1..6)	15	1	1	8	2	20
	1	2	3	4	5	6

d) Zenbaki osoz osatutako A(1..n) bektorea emanda, elementu bikoiti denak A(1..n)-ko ezkerreko aldean eta bakoiti denak eskuineko aldean ipini eta p aldagai boolearrean A(1..n)-ko elementuetatik gutxienez erdiak bikoitiak al diren erabakitzen duen programa dokumentatu zehaztutako puntuetako asertzioak idatziz. Asertzioak idazterakoan aurreko ataletako predikatuak erabili behar dira.

```
{Hasierako baldintza} ≡ {A(1..n) = (a1, a2, …, an) ∧ n ≥ 1}
i : = 1;
k : = n + 1;
while (1) {Inbariantea} i ≠ k loop
    (2) {Tarteko asertzioa}
    if A(i) mod 2 ≠ 0 then (3) {Tarteko asertzioa}
                           lag : = A(i);
                           (4) {Tarteko asertzioa}
                           A(i) : = A(k – 1);
                           A(k – 1) : = lag;
                            (5) {Tarteko asertzioa}
                           k : = k – 1;
    else (6) {Tarteko asertzioa}
        i : = i + 1;
    end if;
end loop;
(7) {Tarteko asertzioa}
p : = ((i – 1) > n / 2);
(8) {Bukaerako baldintza}
(9) {E espresioa}
```

Soluzioa:

a) **bikoitia(x)** ≡ {x mod 2 = 0}

b) **partizioa(C(1..r), x, z)** ≡ {∀i (1 ≤ i ≤ x – 1 → bikoitia(C(i))) ∧
∧ ∀i (z ≤ i ≤ r → ¬bikoitia(C(i)))}

c) **perm(C(1..r), (c_1, c_2, …, c_r))** ≡
≡ {∀i (1 ≤ i ≤ r → Nj (1 ≤ j ≤ r ∧ C(i) = C(j)) = Nk (1 ≤ k ≤ r ∧ C(i) = c_k))

d) Hasteko hasierako baldintza, bukaerako baldintza (8), while-aren ondoren dagoen asertzioa (7) eta inbariantea (1) eman beharko dira. Kasu honetan hasierako baldintza enuntziatuan emana dator eta ondorioz beste hirurak eman beharko dira:

(8) {perm(A(1..n), (a_1, a_2, …, a_n)) ∧ ∃j (1 ≤ j ≤ n + 1 → partizioa(A(1..n), j, j)) ∧
p ↔ (Nj (1 ≤ j ≤ n ∧ bikoitia(A(j))) > (n / 2))}

(7) {(1 ≤ i ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ partizioa(A(1..n), i, i)}

(1) {(1 ≤ i ≤ k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ partizioa(A(1..n), i, k)}

Jarraian while-aren barruan dauden asertzioak emango dira. While-aren barruko asertzioak ematean inbariantearekiko edo asertzio bakoitzaren aurreko asertzioarekiko zer aldatzen den zehaztu beharko da:

(2) {(1 ≤ **i < k** ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ partizioa(A(1..n), i, k)}

(2) zenbakiari dagokion lekuan gaudenean badakigu while-ean sartu garela eta beraz baldintza bete egin dela. Ondorioz i ≠ k betetzen dela ziurtatu dezakegu eta hori da (2) puntuan (1) puntuarekiko aldatzen dena.

(3) {(1 ≤ i < k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ partizioa(A(1..n), i, k)
∧ **¬bikoitia(A(i))**}

(3) zenbakiari dagokion puntuan gaudenean badakigu if-eko then kasuan sartu garela eta if-eko baldintza bete egiten dela. Beraz badakigu A(i) ez dela bikoitia eta hori da (3) puntuan (2) puntuan esaten denari gehitu diezaiokeguna.

(4) {(1 ≤ i < k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧ partizioa(A(1..n), i, k)
∧ ¬bikoitia(A(i)) ∧ **lag = A(i)**}

(4) puntuan gaudenean badakigu if-eko then kasutik joan garela eta lag aldagaiari A(i) esleitu zaiola. Beraz (4) puntuko asertzioa (3) puntuko asertzioari informazio hori gehituz lortu da.

(5) {(1 ≤ i < k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧
partizioa(A(1..n), i, **k – 1**)}

(5) zenbakia dagoen lekuan gaudenean badakigu if-eko else kasuan sartu garela eta i eta k – 1 posizioetako balioak trukatu egin direla baina momentuz i eta k-ren balioak ez dira eguneratu. (5) puntuko asertzioa (4) puntuko asertzioa aldatuz lortu da eta egindako aldaketa elementu bakoitiak k – 1 posiziotik aurrera daudela esatea izan da.

(6) {(1 ≤ i < k ≤ n + 1) ∧ perm(A(1..n), (a_1, a_2, …, a_n)) ∧
partizioa(A(1..n), **i + 1**, k)}

(6) zenbakiari dagokion puntuan gaudenean badakigu i posizioko elementua bikoitia dela eta ez dela mugitu behar, egin beharreko bakarra i aldagaia eguneratzea izango da baina momentuz i aldagaia ez denez eguneratu, (6) puntuko asertzioan elementu bikoitia i posizioraino iristen direla esan beharko da. Asertzio hori (2) puntuko asertzioa aldatuz lortu da, (2) puntukoa baita bere aurreko asertzioa.

Bukatzeko E espresioa emango da:

(9) {E = k – i}
Adibide honetan ere inbariantea betetzen den puntuan gauden bakoitzean E espresioak zenbat buelta falta diren adieraziko digu era zehatzean.