```
module Sarrera_irteera where
import Data.List


-------------------------------------
--Aurredefinitutako String mota [Char] motaren baliokidea edo
--sinonimoa da. Beraz String motako elementuak Char motako
--osagaiak dituzten zerrendak dira.
--Char motako elementuak 'a', '?' edo '7' bezala idazten dira.
--Char motako zerrenda bat edo String motako elementu bat
--hiru eratara idatz daiteke:
-- 'A':'z':'a':'r':'o':'a':[]
-- ['A', 'z', 'a', 'r', 'o', 'a']
-- edo
-- "Azaroa"

-------------------------------------
--Mezuak pantailan aurkezteko aurredefinitutako putStr funtzioa daukagu.
--Bere mota honako hau da: putStr:: String -> IO ()
--Beraz, string motako elementu bat edo karaktere-kate bat emanda,
--IO () motako zerbait itzuliko du.
--IO () motako zerbait lortuko dela esaterakoan, sarrera/irteerako
--ekintzaren bat burutuko dela baina emaitzarik ez dela gordeko
--adierazten da.
--Erabiltzeko era: putStr "348"
-- putStr "Azaroa"

--putStr funtzioa erabiltzen duen kaixo izeneko funtzioaren definizioa
--dator jarraian.
--kaixo funtzioak beti mezu bera aurkeztuko du.
--Erabiltzeko era: kaixo

kaixo:: IO ()
kaixo = putStr "Kaixo mundua!!"

-------------------------------------
--Argumentu bezala emandako karaktere-katea aurkeztuko duen aurkeztu
--izeneko funtzioa definituko da jarraian.
--Funtzio honek putStr funtzioak egiten duen gauza bera egingo du.
--Beraz ez du ezer berririk egiten eta putStr funtzioari
--izena aldatzeko bakarrik balio du.

--Erabiltzeko era:
-- aurkeztu "Azaroa"
-- aurkeztu "348"
-- aurkeztu ""
-- aurkeztu "2018ko azaroa"

aurkeztu:: String -> IO ()
aurkeztu kat = putStr kat

-------------------------------------

--Argumentu bezala emandako karaktere-kate bat aurkeztu eta gero
--lerroz aldatzeko aurredefinitutako putStrLn funtzioa daukagu.

--Bere mota honako hau da: putStrLn:: String -> IO ()
--IO () motako zerbait lortuko dela esaterakoan, sarrera/irteerako
--ekintzaren bat burutuko dela baina emaitzarik ez dela gordeko
--adierazten da.

--Erabiltzeko era:
-- putStrLn "Azaroa"
-- putStrLn "348"
-- putStrLn ""
-- putStrLn "2018ko azaroa"

--Jarraian, argumentu bezala emandako karaktere-katea
```

```
--aurkeztu eta hurrengo lerrora jauzia egiten duen
--aurkeztu_eta_jauzi izeneko funtzioa definituko da.

--Funtzio honek putStrLn funtzioak egiten duen gauza bera egingo du.
--Beraz ez du ezer berririk egiten eta putStrLn funtzioari
--izena aldatzeko bakarrik balio du.

--Erabiltzeko era:
-- aurkeztu_eta_jauzi "Azaroa"
-- aurkeztu_eta_jauzi "348"
-- aurkeztu_eta_jauzi ""
-- aurkeztu_eta_jauzi "2018ko azaroa"

aurkeztu_eta_jauzi:: String -> IO ()
aurkeztu_eta_jauzi kat = putStrLn kat

-------------------------------------
-- Lerro aldaketa edo lerro jauzia "\n" bezala adieraz daiteke.
-- "2014ko azaroa"++"\n" ++ "Bilbao" katea aurkezten badugu,
-- 2014ko azaroa eta Bilbao karaktere-kateak lerro desberdinetan
-- agertuko dira.

--Frogatu:
--aurkeztu ("2014ko azaroa"++"\n" ++ "Bilbao")

-------------------------------------
--Orain "2018ko azaroa" eta "Bilbao" karaktere-kateak
--bi lerrotan aurkezteko do notazioa eta putStrLn
--funtzioa erabiliko ditugu.
--Horretarako, ab izeneko funtzioa definituko dugu.
--Funtzio horrek beti "2018ko azaroa" eta "Bilbao"
--karaktere-kateak bi lerrotan aurkeztuko ditu.

ab:: IO ()
ab = do
        putStrLn "2018ko azaroa"
        putStr "Bilbao"

--IO () motako zerbait lortuko dela esaterakoan, sarrera/irteerako
--ekintzaren bat burutuko dela baina emaitzarik ez dela gordeko
--adierazten da.

{-
-------------------------------------

--putStr eta putstrLn funtzioek karaktere-kateak aurkezteko balio
--dute bakarrik. Ondorioz, putStr "348" idazten badugu, 348 aurkeztuko da
--pantailan, baina putStr 348 idazten badugu, errorea sortuko da.

--Zer egin dezakegu orduan zenbakizko balio bat aurkezteko?
--Aurkeztu daitezkeen motetako elementuak karaktere-kate bihurtzeko
--balio duen show funtzio aurredefinitua dauka Haskell-ek.

--show funtzioaren mota honako hau da:
--show:: Show t => t -> String

--Beraz, 348 balioa aurkezteko bi aukera ditugu orain:
-- putStr "348"
--edo
-- putStr (show 348)

--show funtzioak 348 zenbakia "348" karaktere-katera bihurtzen du.

-}


-------------------------------------
--Adibide honetan show erabiltzea nahitaezkoa da:
```

```
batura_aurkeztu:: Int -> Int -> IO ()
batura_aurkeztu x y = putStr ((show x) ++ " + " ++ (show y) ++ " = " ++ (show (x
+ y)))

-------------------------------------

--Adibide honetan ere show beharrezkoa da.
--Aurreko funtzioarekin alderatuz, mezu desberdina aurkeztuko da.

batura_aurkeztu2 :: Int -> Int -> IO ()
batura_aurkeztu2 x y = putStr ((show x) ++ " gehi " ++ (show y) ++ " " ++ (show
(x + y)) ++ " da")

-------------------------------------


--Orain batura_aurkeztu2 funtzioak egiten duen gauza bera do
--notazioa erabiliz egingo da.

--Sarrera/irteera-ko ekintzak daudenean, ekintza horiek
--zein ordenetan burutu behar diren adierazteko do notazioa erabiltzen da.

batura_aurkeztu3 :: Int -> Int -> IO ()
batura_aurkeztu3 x y = do
                       putStr (show x)
                       putStr " gehi "
                       putStr (show y)
                       putStr " "
                       putStr (show (x + y))
                       putStr " da"
-------------------------------------



--Hotel bati buruzko informazioa emanda (eraikitze-data,
--gela-kopurua eta jatetxerik ba al duen ala ez),
-- informazio hori aurkeztuko du jarraian datorren
--funtzioak do notazioa erabiliz.

hotela :: Int -> Int -> Bool -> IO ()
hotela e_data gela_kop  jatetxea = do
                                     putStr "Eraikitze-data: "
                                     putStrLn (show e_data)
                                     putStr "Gela-kopurua: "
                                     putStrLn (show gela_kop)
                                     putStr "Jatetxea badu: "
                                     putStrLn (show jatetxea)

-------------------------------------

--hotela2 izeneko funtzioak hotela funtzioak egiten duen gauza
--bera egiten du, hau da, hotel bati buruzko informazioa emanda
--(eraikitze-data, gela-kopurua eta jatetxerik ba al duen ala ez),
-- informazio hori aurkeztuko du pantailan do notazioa erabiliz.

--Baina hirugarren parametroaren kasuan "Bai" edo "Ez" aurkeztuko
--da True edo False aurkeztu beharrean.

--Horretarako if-then-else egitura erabiliko da.
--Orain arte funtzioak definitzerakoan | notazioa erabili dugu
--kasu desberdinak bereizteko, baina batzutan if-then-else
--egitura hobea da, batez ere do notazioaren barruan.


hotela2 :: Int -> Int -> Bool -> IO ()
hotela2 e_data gela_kop  jatetxea = do
                                      putStr "Eraikitze-data: "
```

```
                                putStrLn (show e_data)
                                putStr "Gela-kopurua: "
                                putStrLn (show gela_kop)
                                putStr "Jatetxea badu: "
                                if jatetxea
                                    then putStrLn "Bai"
                                    else putStrLn "Ez"

-------------------------------------

--hotela3 izeneko funtzioak hotela eta hotela2 funtzioek egiten
--duten gauza bera egiten du, hau da, hotel bati buruzko
--informazioa emanda (eraikitze-data, gela-kopurua eta jatetxerik
--ba al duen ala ez), informazio hori aurkeztuko du pantailan
--do notazioa erabiliz.

--Baina kasu honetan datu batzuk aurkezteko print
--funtzioa erabiliko da.

--print funtzioaren mota honako hau da:
-- print:: Show t => t -> IO ()
-- eta putStrLn eta show funtzioak batera erabiliz lortzen
--dena burutzeko balio du.
--Beraz, putStrLn (show e_data) eta print e_data
--gauza bera dira.

hotela3 :: Int -> Int -> Bool -> IO ()
hotela3 e_data gela_kop  jatetxea = do
                                putStr "Eraikitze-data: "
                                print e_data
                                putStr "Gela-kopurua: "
                                print gela_kop
                                putStr "Jatetxea badu: "
                                if jatetxea
                                    then putStrLn "Bai"
                                    else putStrLn "Ez"

-------------------------------------
--Orain datorren funtzioan pantailatik lerro oso bat irakurriko da.
--Horretarako, aurredefinitutako getLine funtzioa erabiliko da.

--getLine funtzioaren mota honako hau da:
--getLine:: IO String

--IO String motaren bidez, sarrera/irteerako ekintzaren bat burutu dela
--eta gero beste funtzioren batek erabili ahal izango duen String
--motako datu bat gorde dela adierazten da.

jaso_eta_aurkeztu :: IO ()
jaso_eta_aurkeztu = do putStrLn "Lerro batean testua idatzi: "
                       lerroa <- getLine
                       putStrLn ("Hau da jasotako testua: " ++ lerroa)


--Definizio horretan <- eragilea agertzen da. Eragile hori ez da esleipena
--Eragile horrek IO t motako elementu bat hartu eta t motako osagaia
--ateratzen du. Beraz, <- eragileak IO t erako elementu bati IO zatia edo
--bilgarria kentzen dio.

{-Jarraian datorren definizioa ez da zuzena, izan ere bertan <- eragilea
--esleipena balitz bezala erabiltzen da, baina <- ez da esleipena,
--IO bilgarria kentzeko balio du.

fff :: IO ()
fff = do putStrLn "AAA"
         lerroa <- "asd"
         putStrLn ("Testua hau da: " ++ lerroa)
```

```
--Honako funtzio hau ere ez da zuzena, 4 balioak ez duelako IO
--bilgarria.

fff2 :: IO ()
fff2 = do putStrLn "AAA"
          lerroa <- 4
          putStr ("Testua hau da: ")
          print lerroa

--Bestalde, return funtzioak elementu bat IO bilgarriarekin biltzeko
--balio du, hau da, t motako elementu bat hartuta, IO t motako
--elementu bat lortzeko balio du.
return :: t -> IO t

Jarraian datorren ggg funtzioaren bidez ikus daiteke hori:
-}
-------------------------------------

ggg :: IO ()
ggg = do putStrLn "AAA"
         lerroa <- return "asd"
         putStrLn ("Testua hau da: " ++ lerroa)

-------------------------------------

ggg2 :: IO ()
ggg2 = do putStrLn "AAA"
          lerroa <- return 4
          putStr ("Testua hau da: ")
          print lerroa

-------------------------------------

batu_si :: IO ()
batu_si = do putStrLn "Bi zenbaki idatzi, bakoitza lerro batean:"
             z1 <- lortu_zenb
             z2 <- lortu_zenb
             print (z1 + z2)
-------------------------------------

batu_si2 :: IO ()
batu_si2 = do putStrLn "Bi zenbaki idatzi, bakoitza lerro batean:"
              z1 <- getLine
              z2 <- getLine
              let y1 = (read z1 :: Int)
              let y2 = (read z2 :: Int)
              print (y1 + y2)
-------------------------------------
--read funtzioak karaktere-kate bat beste motako elementu bat
--bihurtzeko balio du

lortu_zenb :: IO Int
lortu_zenb = do z <- getLine
                return (read z :: Int)

-------------------------------------
--Funtzio honek Int motako zerrenda bat lortuko du

lortu_zerrenda :: IO [Int]
lortu_zerrenda = do z <- getLine
                    return (read z :: [Int])
-------------------------------------
--Funtzio honek Int motako zerrenda bat lortu eta
--bertako zenbakien batura aurkeztuko du pantailan

batu_zerrenda :: IO ()
batu_zerrenda = do putStrLn "Zerrenda bat idatzi:"
                   zer <- lortu_zerrenda
```

```
                print (sum zer)

-------------------------------------
--Funtzio honek bi zenbaki oso eskatu, jaso, batura kalkulatu
--eta aurkeztu eta jarraian prozesua behin eta berriz
--errepikatuko du  amaierarik gabe.

--Programa bukatzeko konpiladoretik edo Haskell menutik
--gelditu beharko da. Beraz, alde horretatik ez da oso egokia.

batu_si2_sek :: IO ()
batu_si2_sek = do putStrLn "Bi zenbaki idatzi, bakoitza lerro batean:"
                  z1 <- getLine
                  z2 <- getLine
                  let y1 = (read z1 :: Int)
                  let y2 = (read z2 :: Int)
                  print (y1 + y2)
                  batu_si2_sek
-------------------------------------
--Funtzio honek bi zenbaki oso eskatu, jaso, batura kalkulatu
--eta aurkeztu eta jarraian prozesua behin eta berriz
--errepikatuko du  amaierarik gabe.

--Programa bukatzeko konpiladoretik edo Haskell menutik
--gelditu behgarko da. Beraz, alde horretatik ez da oso egokia.

--batu_si2_sek funtzioak egiten duen gauza bera egiten du baina
--emaitza aurkezterakoan mezu desberdina erabiltzen da.


batu_si3_sek :: IO ()
batu_si3_sek = do putStrLn "Bi zenbaki idatzi, bakoitza lerro batean:"
                  z1 <- getLine
                  z2 <- getLine
                  let y1 = (read z1 :: Int)
                  let y2 = (read z2 :: Int)
                  putStr ((show y1) ++ " + " ++ (show y2) ++ " = ")
                  print (y1 + y2)
                  batu_si3_sek
-------------------------------------
--Funtzio honek bi zenbaki oso eskatu, jaso, batura kalkulatu
--eta aurkeztu eta jarraian prozesua behin eta berriz
--errepikatuko du erabiltzaileak bukatzea nahi duela esan arte.

--Beraz programa bukatzeko era egokiagoa da.

--Hala ere, erabiltzaileari jarraitzea nahi al duen galdetutakoan,
--erabiltzaileak b (bai) edo e (ez) ez badu erantzuten ere e erantzun
--duela ulertuko da. Beraz, erantzuna ez da kontrolatzen eta
--alde horretatik ez da guztiz egokia.

--Funtzio honetan erabiltzailearen erantzuna String motako elementu
--bezala mantentzen da eta horregatik "b" (komatxo bikoitzekin)
--erantzun al duen aztertuko da.

batu_si_sek_buk :: IO ()
batu_si_sek_buk = do putStrLn "Bi zenbaki idatzi, bakoitza lerro batean:"
                     z1 <- getLine
                     z2 <- getLine
                     let y1 = (read z1 :: Int)
                     let y2 = (read z2 :: Int)
                     putStr ((show y1) ++ " + " ++ (show y2) ++ " = ")
                     print (y1 + y2)
                     putStrLn "Beste batuketarik? (b/e): "
                     besterik <- getLine
                     if besterik == "b"
                        then batu_si_sek_buk
                        else putStrLn "Bukaera."
```

```
-------------------------------------
--Funtzio honek batu_si_sek_buk funtzioak egiten duen gauza bera
--egiten du eta erabiltzailearen b edo e erantzunarekin
--arazo bera dauka.

--Aldaketa bakarra honako hau da: funtzio honetan erabiltzailearen
--erantzuna String motako elementu bezala mantendu beharrean, Char
--motako bezala kontsideratzen da da eta horregatik 'b' (komatxo
--sinpleekin edo apostrofoarekin) erantzun al duen aztertuko da.


batu_si_sek_buk2 :: IO ()
batu_si_sek_buk2 = do putStrLn "Bi zenbaki idatzi, bakoitza lerro batean:"
                      z1 <- getLine
                      z2 <- getLine
                      let y1 = (read z1 :: Int)
                      let y2 = (read z2 :: Int)
                      putStr ((show y1) ++ " + " ++ (show y2) ++ " = ")
                      print (y1 + y2)
                      putStrLn "Beste batuketarik? (b/e): "
                      besterik <- getLine
                      let erantzuna = head besterik
                      if erantzuna == 'b'
                         then batu_si_sek_buk2
                         else putStrLn "Bukera."
-------------------------------------
--Funtzio honek batu_si_sek_buk eta batu_si_sek_buk2 funtzioek
--egiten dutenaren antzekoa egiten du, baina hemen erabiltzaileak
--derrigorrez b (bai) edo e (ez) erantzun beharko du, ez da
--beste erantzunik onartuko.

--Funtzio honen beste ezaugarri garrantzitsu bat honako hau da:
--Erabiltzaileak gutxienez batuketa bat burutu beharko duela.


batu_si_sek_buk3 :: IO ()
batu_si_sek_buk3 = do putStrLn "Bi zenbaki idatzi, bakoitza lerro batean:"
                      z1 <- getLine
                      z2 <- getLine
                      let y1 = (read z1 :: Int)
                      let y2 = (read z2 :: Int)
                      putStr ((show y1) ++ " + " ++ (show y2) ++ " = ")
                      print (y1 + y2)
                      erantzuna <- erantzuna_eskatu
                      if erantzuna == 'b'
                         then batu_si_sek_buk3
                         else putStrLn "Bukaera."
-------------------------------------
--Funtzio honek erabiltzaileari b (bai) edo e (ez) erantzuteko
--eskatzen dio eta erabiltzaileak b edo e erantzun arte
--errepikatuko du eskatze-prozesua. Ez du beste erantzunik
--onartuko.


erantzuna_eskatu:: IO Char
erantzuna_eskatu = do putStrLn "beste batuketarik? (b/e): "
                      besterik <- getLine
                      let er = head besterik
                      if er `elem` "be"
                        then (return er)
                        else do
                               putStrLn "Erantzuna ez da egokia..."
                               erantzuna_eskatu
-------------------------------------
--Funtzio honek aurrekoak egiten duen gauza bera egiten du
--baina mezu desberdina aurkezten du hasieran,
--funtzio desberdin batean erabiliko delako.
```

```
erantzuna_eskatu2:: IO Char
erantzuna_eskatu2 = do putStrLn "Bi zenbaki batzerik nahi al duzu? (b/e): "
                       besterik <- getLine
                       let er = head besterik
                       if er `elem` "be"
                          then (return er)
                          else do
                                 putStrLn "Erantzuna ez da egokia..."
                                 erantzuna_eskatu2

-------------------------------------

--Funtzio honek batu_si_sek_buk3 funtzioak
--egiten duenaren antzekoa egiten du, baina hemen erabiltzaileak
--bukatzea aukera dezake batuketa bat bera ere egin gabe.

batu_si_sek_buk4 :: IO ()
batu_si_sek_buk4 = do erantzuna <- erantzuna_eskatu2
                      if erantzuna == 'b'
                         then do
                                putStrLn "Bi zenbaki idatzi, bakoitza lerro
batean:"
                                z1 <- getLine
                                z2 <- getLine
                                let y1 = (read z1 :: Int)
                                let y2 = (read z2 :: Int)
                                putStr ((show y1) ++ " + " ++ (show y2) ++ " =
")
                                print (y1 + y2)
                                batu_si_sek_buk4
                           else putStrLn "Bukaera."
-------------------------------------
```