94 (υποδειγματικά) Λυμένες Ασκήσεις Α.Ε.Π.Π.

(αρκετές απ' αυτές για δυνατούς μαθητές)

version 0.1

Θεσσαλονίκη Ιανουάριος 2014

Απόστολος Δεμερτζής

Πρόλογος

Οι ασκήσεις αυτές δεν αποτελούν διδακτικό βοήθημα για τον μαθητή που μαθαίνει ακόμη τις εντολές και τους κανόνες. Δεν προσφέρουν μεθοδολογία, ούτε κατηγοριοποιούν τις ασκήσεις. Σε καμιά περίπτωση δεν είναι προτεινόμενα θέματα εξετάσεων. Αποκλειστικός σκοπός τους είναι να δώσουν στους υποψήφιους την δυνατότητα να δουν έτοιμο κώδικα που λύνει ενδιαφέροντα προβλήματα. Η αφορμή για την συγγραφή των σημειώσεων ήταν ακριβώς αυτή. Ένα δείγμα καλογραμμένου κώδικα για κάποιον που έρχεται για πρώτη φορά σε επαφή με τον προγραμματισμό. Αυτό σημαίνει ότι πρέπει να κατέχεται όλη την ύλη πριν αρχίσετε να διαβάζετε.

Δυστυχώς οι λυμένες ασκήσεις και τα παραδείγματα του βιβλίου είναι λίγα και τα περισσότερα θυμίζουν λίγο νηπιαγωγείο. Στα διάφορα βοηθήματα του εμπορίου μπορεί να βρει κανείς εξαιρετικές ασκήσεις, αλλά αυτές είναι θαμμένες κάτω από τόνους πανομοιότυπων, μέτριων προβλημάτων, λυμένων με τον ίδιο κάθε φορά τρόπο. Ο καλός μαθητής βαριέται. Και πρέπει να διαβάσει εκατοντάδες σελίδες για να βρει μερικά ενδιαφέροντα κομμάτια κώδικα. Ο όγκος των βιβλίων είναι αποτρεπτικός. Στην προσπάθειά μας (οι καθηγητές) να καλύψουμε κάθε πιθανή και απίθανη εκδοχή των ίδιων και ίδιων ασκήσεων (να πιάσουμε τα θέματα) χάνουμε την ουσία του προγραμματισμού. Προγραμματισμός σημαίνει να μάθεις να σκέφτεσαι. Αν μάθεις να σκέφτεσαι, μετά οι τεχνικές και τα κολπάκια χωράν μέσα σε λίγες δεκάδες σελίδες. Δεν χρειάζονται τόμους. Λοιπόν, υπόσχομαι στους δυνατούς μαθητές ότι δεν θα βαρεθούν διαβάζοντας αυτές τις σημειώσεις.

Δεν έχω καμία εμπειρία από την διδασκαλία του μαθήματος στην τάξη. Δίδαξα για μία χρονιά το μάθημα, πριν από αρκετά χρόνια. Αυτό ήταν όλο. Η μόνη μου εμπειρία είναι η κόρη μου, υποψήφια την τρέχουσα χρονιά, στην οποία προσπάθησα να διδάξω προγραμματισμό. Οι ασκήσεις είναι αυτές που λύναμε μαζί κατά την διάρκεια των τελευταίων έξι μηνών. Σαν τελευταία επανάληψη προσπάθησα να τις συγκεντρώσω μαζί με τις λύσεις. Κάποιες απ' τις λύσεις χαθήκαν, κάποιες δεν ήταν τόσο καλογραμμένες. Έτσι αποφάσισα να ξαναγράψω τον κώδικα από την αρχή. Πρόσθεσα μερικά σχόλια και συμβουλές και έτσι κατέληξα στις σημειώσεις που κρατάτε στα χέρια σας. Ουσιαστικά οι σημειώσεις αυτές είναι γραμμένες για ένα μόνο παιδί, την κόρη μου.

Το τελικό αποτέλεσμα ήταν πολύ καλό για να το κρατήσω για τον εαυτό μου. Αποφάσισα λοιπόν να το επιμεληθώ λίγο παραπάνω, φτιάχνοντας δυο τρία σχήματα και να το κοινοποιήσω (στο Στέκι των Πληροφορικών). Οι ασκήσεις είναι κλεμμένες από παντού. Δεν μπορώ να θυμηθώ από που προέρχεται η κάθε μία. Σίγουρα προέρχονται όλες από αναζήτηση στο Διαδίκτυο. Δυο τρεις είναι παρμένες από το project Euler και άλλες τόσες από το dzone.com και την στήλη Thursday code puzzler. Οι περισσότερες ασκήσεις είναι διάσημες και μπορεί να τις βρει κανείς σε πολλά βιβλία. Για τις υπόλοιπες δεν θα διακινδυνεύσω να αναφέρω την προέλευση τους, επειδή ειλικρινά δεν θυμάμαι. Αν κάποιος αναγνωρίσει δική

του άσκηση, ας με ενημερώσει να περιλάβω την πηγή (η να αποσύρω την άσκηση). Σε κάθε περίπτωση, οι σημειώσεις αφορούν τον κώδικα και όχι τις ασκήσεις.

Φυσικά, εφόσον οι ασκήσεις είναι κλεμμένες, επιτρέπεται να κλέψετε και εσείς ότι θέλετε (με την προϋπόθεση ότι δεν έχει αντίρρηση ο αρχικός δημιουργός). Ο κώδικας των λύσεων είναι συνηθισμένος και μπορεί να τον βρει κανείς σε οποιοδήποτε εισαγωγικό βιβλίο προγραμματισμού. Δεν διεκδικώ την πατρότητα καμιάς λύσης. Το παρόν είναι ελεύθερο να το κάνετε ότι θέλετε (ακόμη και να το πουλήσετε).

Θα κλείσω τον μακροσκελή πρόλογο με δυο λόγια προς τους υποψήφιους των εξετάσεων που διαβάζουν ΑΕΠΠ. Οι ασκήσεις εδώ μέσα δεν αποτελούν τυπικό δείγμα ασκήσεων ΑΕΠΠ. Μόνο το 1/3 απ' αυτές θα μπορούσατε να λύσετε μόνοι σας. Άλλο 1/3 είναι αρκετά δύσκολες ασκήσεις, μέσα στις δυνατότητες των καλών μαθητών. Τέλος, το υπόλοιπο 1/3 είναι ασκήσεις για λίγους. Αν αισθανθείτε άβολα διαβάζοντας τες, αν πιστεύετε ότι είναι υπερβολικές και παρατραβηγμένες για την ΑΕΠΠ, τότε ναι έχετε δίκιο. Τουλάχιστον το 1/3 απ' αυτές είναι και υπερβολικές και παρατραβηγμένες. Ο σκοπός τους όμως δεν είναι να τις λύσετε. Ο σκοπός είναι να διαβάσετε την λύση. Διαβάστε μόνο ότι σας κάνει να αισθάνεστε άνετα και προσπεράστε ότι δεν καταλαβαίνετε. Οι ασκήσεις αυτές είναι "ειδικού σκοπού". Σε καμιά περίπτωση δεν χάνεται τίποτα προσπερνώντας τα δύσκολα κομμάτια (ή και όλο το κείμενο). Θα τα πάτε περίφημα στις εξετάσεις και χωρίς αυτές τις ασκήσεις.

Αν όμως διαβάσετε και τις δύσκολες ασκήσεις και νιώσετε ότι είναι το στοιχείο σας, αν καταλαβαίνετε τι κάνει η κάθε εντολή, αν συγκινηθείτε από ένα κομψό κομμάτι κώδικα ή από μια έξυπνη λύση, αν δεν μπορείτε να σταματήσετε να διαβάζετε την μία άσκηση μετά την άλλη, τότε αυτές οι ασκήσεις είναι για σας. Ο σκοπός τους είναι να δείξουν στον σημερινό 17χρονο με τι ασχολείται ο προγραμματισμός. Υπάρχουν εκατοντάδες προβλήματα σαν αυτά, τα οποία αντιμετωπίζουν οι προγραμματιστές κάθε μέρα, ακόμη και στην ανάπτυξη πολύ απλών προγραμμάτων. Ίσως θα πρέπει να σκεφτείτε την περίπτωση να γίνετε επαγγελματίας προγραμματιστής. Ακόμη όμως κι αν ασχοληθείτε με κάτι άλλο στη ζωή σας, μπορείτε να μάθετε προγραμματισμό ερασιτεχνικά. Υπάρχουν πολλά επαγγέλματα στα οποία θα σας φανεί χρήσιμος, πέρα από την ευχαρίστηση που προσφέρει η ίδια η εκμάθηση.

Ο κώδικας των λύσεων γράφηκε σε πολύ μικρό χρονικό διάστημα. Δεν είναι εξαντλητικά ελεγμένος. Ζητώ συγνώμη, εκ των προτέρων, για τα λάθη είτε στον κώδικα είτε στο κείμενο. Η εμπειρία μου λέει ότι θα πρέπει να υπάρχουν αρκετά. Αν βρείτε κάτι, σημαντικό ή ασήμαντο, θα ήθελα να το ξέρω.

Για ενστάσεις, προτάσεις, διορθώσεις, μπινελίκια, ή απλώς επικοινωνία το ηλεκτρονικό ταχυδρομείο μου είναι:

apoldem@gmail.com

Απόστολος Δεμερτζής

αυτός ο κόπος ανήκει στην Ηλιάνα

Πίνακας περιεχομένων

Δομή Ακολουθίας

Άσκηση 1 - απόλυτη τιμή

Να γραφεί αλγόριθμος που να ζητά έναν αριθμό και να υπολογίζει την απόλυτη τιμή του αριθμού

Απάντηση

```
Αλγόριθμος ΑπόλυτηΤιμή
Εκτύπωσε "Δώστε έναν αριθμό"
Διάβασε α

Αν α < 0 τότε α ←  (-1)*α

Εκτύπωσε "Η απόλυτη τιμή είναι : ", α
Τέλος
```

Άσκηση 2

Ένας παντρεμένος υπάλληλος έχει έναν βασικό μισθό. Παίρνει επιπλέον 35€ επίδομα γάμου και 20€ επίδομα για κάθε παιδί. Επί του βασικού μισθού έχει κρατήσεις 20% προς το ασφαλιστικό του ταμείο. Στο ποσό που απομένει μετά την αφαίρεση των ασφαλιστικών εισφορών γίνεται παρακράτηση 11% για προκαταβολή φόρου.

Να αναπτύξετε αλγόριθμο που να ρωτάει τον βασικό μισθό και τον αριθμό παιδιών και να εμφανίζει τις συνολικές ακαθάριστες αποδοχές, τις κρατήσεις και τέλος το καθαρό ποσό που θα εισπράξει ο υπάλληλος.

Απάντηση

```
Αλγόριθμος Αποδοχές
Δεδομένα // μισθός, παιδιά //
 ! έσοδα
επίδομα ← 35 + παιδιά*20
ακαθάριστα ← μισθός + επίδομα

 ! έξοδα
ασφάλεια ← μισθός* 0.2
αποδοχέςΠροΦόρου ← ακαθάριστα - ασφάλεια
φόρος ← αποδοχέςΠροΦόρου* 0.11

 ! εκκαθάριση
καθαρά ← ακαθάριστα - ασφάλεια - φόρος

Εμφάνισε "Ακαθάριστος Μισθός : ", ακαθάριστα
Εμφάνισε "Κρατήσεις : ", ασφάλεια + φόρος
Εμφάνισε "Καθαρός μισθός : ", καθαρά

Τέλος
```

Άσκηση 3 - αντιμετάθεση

Να διαβαστούν δύο πραγματικοί αριθμοί, οι οποίοι να εκχωρηθούν σε δύο μεταβλητές α και β. Στη συνέχεια να γίνει ανταλλαγή των τιμών τους. Να μην χρησιμοποιηθεί η ενσωματωμένη εντολή της ψευδογλώσσας.

Απάντηση

```
Αλγόριθμος Αντιμετάθεση
Εκτύπωσε "Δώστε τον 1ο αριθμό (α) "
Διάβασε α
Εκτύπωσε "Δώστε τον 2ο αριθμό (β) "
Διάβασε β

προσ ← α     ! ο α κρατιέται σε προσωρινή θέση μνήμης
α ← β        ! τώρα μπορούμε να αλλάξουμε την τιμή του α
β ← προσ     ! το β πρέπει να πάρει την παλιά τιμή του α

Εκτύπωσε "α = ", α
Εκτύπωσε "β = ", β
Τέλος
```

Άσκηση 4 - πόντοι σε σούπερ μάρκετ

Ένα σούπερ μάρκετ κάνει προσφορές στους πελάτες του ανάλογα με τους πόντους που συγκεντρώνουν στις αγορές τους. Για κάθε 3€ αγορών ο πελάτης κερδίζει έναν πόντο. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει το ποσό των αγορών σε € (με δύο δεκαδικά) και να τυπώνει πόσοι πόντοι τού αναλογούν. Παραδείγματα: αν δώσουμε 125.25 μας επιστρέφει 41, αν δώσουμε 56.23 μας επιστρέφει 18, αν δώσουμε 2.5 μας επιστρέφει 0.

Απάντηση

```
Αλγόριθμος ΠόντοιΣούπερΜάρκετ
Εκτύπωσε "Ποιο είναι το ποσό των αγορών? "
Διάβασε ποσό

ποσό_ακέραιο ←  Α_Μ(ποσό)
πόντοι ←  ποσό_ακέραιο div 3
! πόντοι ←  Α_Μ(ποσό) div 3    ! ακόμη καλύτερα σε μια γραμμή

Εκτύπωσε  "οι πόντοι είναι ", πόντοι
Τέλος
```

Άσκηση 5 - σειριακοί υπολογισμοί

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει το ημερομίσθιο ενός εργάτη σε €, τις ημέρες του μήνα που δούλεψε καθώς και τις υπερωρίες του σε ώρες και να υπολογίζει τις ακαθάριστες αποδοχές του, τις κρατήσεις και τέλος τις καθαρές αποδοχές του μήνα. Οι υπερωρίες πληρώνονται με το 20% του ημερομισθίου και οι κρατήσεις του είναι 5% επί των συνολικών αποδοχών του. Παραδείγματα: αν δώσουμε ημερομίσθιο 56€, ημέρες εργασίας 12 και υπερωρίες 9 ώρες, πρέπει να μας επιστρέψει: ακαθάριστες αποδοχές = 772,8€ , κρατήσεις = 38,64€ , καθαρές αποδοχές = 734,16€

Απάντηση

```
Αλγόριθμος ΚαθαρέςΑποδοχές
  Εκτύπωσε "Ποιο είναι το ημερομίσθιο ? "
  Διάβασε ημερομίσθιο
  Εκτύπωσε "Πόσες ήμερες εργασίας μέσα στον μήνα ? "
  Διάβασε ημέρες
  Εκτύπωσε "Πόσες υπερωρίες (ώρες) ; "
  Διάβασε υπερωρίες

  ακαθάριστα ← (ημέρες + υπερωρίες*0.2) * ημερομίσθιο
  !ακαθάριστα ← ημέρες*ημερομίσθιο + υπερωρίες*0.2*ημερομίσθιο (*ΛΑΘΟΣ*)
```

```
  κρατήσεις ← 0.05*ακαθάριστα
  καθαρά ← ακαθάριστα - κρατήσεις

  Εκτύπωσε "Οι ακαθάριστες αποδοχές είναι : ", ακαθάριστα
  Εκτύπωσε "Οι κρατήσεις είναι : ", κρατήσεις
  Εκτύπωσε "Οι καθαρές αποδοχές είναι : ", καθαρά
Τέλος
```

Άσκηση 6 - στρογγυλοποίηση στο 100

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει έναν ακέραιο αριθμό και να τον στρογγυλοποιεί στα δύο τελευταία ψηφία του. Για παράδειγμα: αν δώσουμε 15732 μας δίνει 15700 , αν δώσουμε 15784 μας δίνει 15800 , αν δώσουμε 60 μας δίνει 100 , αν δώσουμε 530 μας δίνει 500 , αν δώσουμε 17 μας δίνει 0.

Απάντηση

```
Αλγόριθμος ΣτρογγυλοποιησηΣτο100
  Εμφάνισε "Δώστε έναν ακέραιο : "
  Διάβασε ακερ

  ακερ ← ακερ + 50    ! προσθέτουμε 50 για την στρογγυλοποίηση
  πραγμ ← ακερ / 100  ! μετακινούμε την υποδιαστολή 2 θέσεις αριστερά
  ακερ ← A_M(πραγμ)   ! κόβουμε τα δεκαδικά
  ακερ ← ακερ*100     ! ξαναγυρνάμε την υποδιαστολή στην θέση της

  Εμφάνισε ακερ
  !Εμφάνισε A_M( (ακερ + 50)/100 ) * 100 ! ο αλγόριθμος σε μία γραμμή
Τέλος
```

Δομή Επιλογής

Άσκηση 7 - τριώνυμο

Να γραφεί αλγόριθμος που να βρίσκει τις ρίζες ενός τριωνύμου. Αρχικά να ρωτά τους συντελεστές α, β και γ του τριωνύμου και κατόπιν να εμφανίζει τις ρίζες ή την ένδειξη “Αδύνατη εξίσωση” αν αυτή δεν έχει πραγματικές ρίζες

Απάντηση

```
Αλγόριθμος Τριώνυμο
Εκτύπωσε "Δώστε τον συντελεστή του μεγιστοβάθμιου όρου (α) : "
Διάβασε α
Εκτύπωσε "Δώστε τον συντελεστής του x (β) : "
Διάβασε β
Εκτύπωσε "Δώστε τον σταθερό όρο (γ) : "
Διάβασε γ

Δ ← β^2 -4*α*γ        ! η διακρίνουσα

Αν α = 0 τότε
  Εμφάνισε "Δεν είναι τριώνυμο"
Αλλιώς_Αν Δ < 0 τότε
  Εμφάνισε "Αδύνατη εξίσωση"
αλλιώς
  Εμφάνισε "1η ρίζα = ", (-β + Τ_Ρ(Δ)) / (2*α)
  Εμφάνισε "2η ρίζα = ", (-β - Τ_Ρ(Δ)) / (2*α)
Τέλος_αν
Τέλος
```

Άσκηση 8 - ελάχιστο/μέγιστο

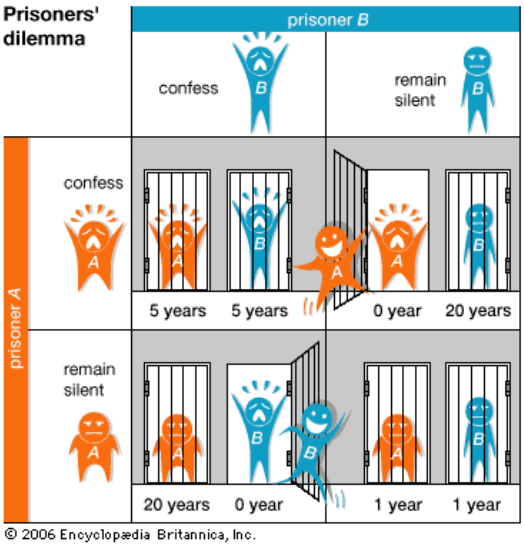
Να γραφεί αλγόριθμος που να ζητά διαδοχικά τρεις αριθμούς και στο τέλος να εμφανίζει τον μικρότερο τους.

Απάντηση

Ο κλασικός αλγόριθμος υπολογισμού μικρότερου ή μεγαλύτερου από μια σειρά αριθμών. Αρχικά θεωρούμε μικρότερο (μεγαλύτερο) τον πρώτο αριθμό που συναντάμε. Έπειτα, κάθε επόμενο αριθμό τον συγκρίνουμε με αυτόν που θεωρούμε εμείς μικρότερο μέχρι εκείνη την στιγμή.

```
Αλγόριθμος Μικρότερος
! στην μεταβλητή μικρ θα κρατάμε τον μικρότερο αριθμό  απ' αυτούς..
! ..που έχουμε συναντήσει μέχρι εκείνη την στιγμή

Εμφάνισε "Δώστε τον 1ο αριθμό : "
Διάβασε α
μικρ ← α                        ! αυτός είναι ο μικρότερος

Εμφάνισε "Δώστε τον 2ο αριθμό : "
Διάβασε β
Αν β < μικρ τότε μικρ ← β       ! αν είναι μικρότερος απ' τον πρώτο

Εμφάνισε "Δώστε τον 3ο αριθμό : "
Διάβασε γ
Αν γ < μικρ τότε μικρ ← γ       ! είναι μικρότερος απ' όλους μέχρι τώρα

Εμφάνισε "μικρότερος είναι ο ", μικρ
Τέλος
```

Άσκηση 9 - το δίλημμα του φυλακισμένου



Το πρόβλημα αυτό είναι γνωστό ως δίλημμα του φυλακισμένου. Η αστυνομία έχει συλλάβει δύο συνεργούς σε αδίκημα. Στον καθένα προτείνεται ο εξής συμβιβασμός:

- Ομολόγησε και πρόδωσε τον συνεργάτη σου όσο αυτός δεν έχει ομολογήσει ακόμη. Αν συμβεί αυτό εσύ θα αφεθείς ελεύθερος, ενώ ο συνεργός σου θα φυλακιστεί για 20 χρόνια.
- Αν ομολογήσει και σε προδώσει αυτός τότε θα αφεθεί αυτός ελεύθερος και θα φυλακιστείς εσύ για 20 χρόνια.
- Αν ομολογήσετε και προδώσετε και οι δύο ταυτόχρονα, τότε θα φυλακιστείτε και οι δύο για 5 χρόνια ο καθένας.
- Αν δεν ομολογήσει κανείς από τους δύο τότε, λόγω έλλειψης στοιχείων, θα καταδικαστείτε και οι δύο για ελαφρύτερα αδικήματα, σε 1 χρόνο ο καθένας.

Το πρόβλημα είναι τι πρέπει να κάνει ο κάθε φυλακισμένος, χωρίς να γνωρίζει τι θα κάνει ο άλλος. Αν μπορούσαν να συνεννοηθούν θα προτιμούσαν να μην προδώσει κανένας και να γλιτώσουν με μικρότερη ποινή. Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να διαβάζει αν ομολόγησαν ή όχι οι δύο συνεργοί ρωτώντας για τον καθένα χωριστά και έπειτα να εμφανίζει τα χρόνια φυλακής που καταδικάστηκε ο καθένας.

Απάντηση

```
Αλγόριθμος ΔίλημμαΦυλακισμένου
Εμφάνισε "Ομολόγησε ο Α (ναι/όχι): "
Διάβασε απάντηση
ομολΑ ← απάντηση = 'ναι'      ! αν ο χρήστης έγραψε 'ναι' τότε Αληθές

Εμφάνισε "Ομολόγησε ο Β (ναι/όχι): "
Διάβασε απάντηση
ομολΒ ← απάντηση = 'ναι'

φυλακήΑ ← 0
φυλακήΒ ← 0
Αν ομολΑ και ομολΒ τότε
  φυλακήΑ ← 5
  φυλακήΒ ← 5
αλλιώς_αν ομολΑ και όχι ομολΒ τότε
  φυλακήΒ ← 20
αλλιώς_αν όχι ομολΑ και ομολΒ τότε
  φυλακήΑ ← 20
αλλιώς
  φυλακήΑ ← 1
  φυλακήΒ ← 1
Τέλος_αν

Εμφάνισε "Ο Α καταδικάστηκε σε ", φυλακήΑ, " χρόνια φυλακή"
Εμφάνισε "Ο Β καταδικάστηκε σε ", φυλακήΒ, " χρόνια  φυλακή"
Τέλος
```

*Αν ξεκινήσατε να λύνετε την άσκηση με φωλιασμένα **Αν** τότε είσαστε λάθος. Η σημερινή επικρατούσα αντίληψη στον προγραμματισμό είναι ότι τα φωλιασμένα **Αν** απαγορεύονται εντελώς. Στο πλαίσιο του ΑΕΠΠ η απαγόρευση δεν μπορεί να είναι τόσο αυστηρή επειδή η γλώσσα δεν μας δίνει την δυνατότητα να βγούμε πρόωρα από ένα μπλοκ κώδικα. Αν οι συνθήκες των **Αν** δεν μπλέκονται μεταξύ τους και η ροή του προγράμματος είναι ξεκάθαρη, τότε μπορούμε να χρησιμοποιήσουμε φωλιασμένα **Αν** (δείτε στην 14 ένα παράδειγμα). Για να αποφύγετε τα φωλιασμένα **Αν** χρησιμοποιήστε λογικές μεταβλητές και σε εξαιρετικές περιπτώσεις σπάστε το πρόγραμμα σε διαδικασίες (δείτε την άσκηση 66).*
Κάντε τον κώδικά σας να διαβάζεται σαν μυθιστόρημα και όχι σαν γρίφος.

Άσκηση 10 - αποφύγετε τα φωλιασμένα Αν (1)

Να γραφεί αλγόριθμος που να ζητά δύο ακέραιους αριθμούς και να ελέγχει αν αυτοί είναι άρτιοι ή περιττοί. Μετά να εμφανίζει το μήνυμα **"και οι δύο άρτιοι"** αν είναι και οι δύο άρτιοι ή **"και οι δύο περιττοί"** αν είναι και οι δύο περιττοί και τέλος **"ένας άρτιος και ένας περιττός"** αν είναι ο ένας άρτιος και ο άλλος περιττός.

Απάντηση

```
Αλγόριθμος ΔυοΆρτιοιΠεριττοί_Αντιπαράδειγμα
Εμφάνισε "Δώστε τον 1ο αριθμό : "
Διάβασε α
Εκτύπωσε "Δώστε τον 2ο αριθμό : "
Διάβασε β

Αν α mod 2 = 0 τότε
  ΑΝ β mod 2 = 0 τότε
```

```
    Εμφάνισε "Και οι δυο άρτιοι"
  αλλιώς
    Εμφάνισε "Ο ένας άρτιος και ο άλλος περιττός"
  Τέλος_αν
αλλιώς
  Αν  β mod 2 = 0 τότε
    Εμφάνισε "Ο ένας άρτιος και ο άλλος περιττός"
  αλλιώς
    Εμφάνισε "Και οι δυο περιττοί"
  Τέλος_αν
Τέλος_αν

Τέλος
```

Τα φωλιασμένα **Αν** πάντα οδηγούν σε ακατανόητους αλγορίθμους. Ο σωστός τρόπος να γράφουμε κώδικα είναι με λογικές μεταβλητές

```
Αλγόριθμος ΔυοΆρτιοιΠεριττοί
Εμφάνισε "Δώστε τον 1ο αριθμό : "
Διάβασε α
Εκτύπωσε "Δώστε τον 2ο αριθμό : "
Διάβασε β

Άρτιος_ο_1ος ←  α mod 2 = 0
Άρτιος_ο_2ος ←  β mod 2 = 0

Αν Άρτιος_ο_1ος και Άρτιος_ο_2ος τότε
  Εμφάνισε "Και οι δυο άρτιοι"
αλλιώς_αν  όχι Άρτιος_ο_1ος και όχι Άρτιος_ο_2ος τότε
  Εμφάνισε "Και οι δυο περιττοί"
αλλιώς
    Εμφάνισε "Ο ένας άρτιος και ο άλλος περιττός"
Τέλος_αν
Τέλος
```

Άσκηση 11 - αποφύγετε τα φωλιασμένα Αν (2)

Ένας προγραμματιστής έγραψε τον παρακάτω απαίσιο κώδικα

```
Αλγόριθμος Θέμα2_2006
Διάβασε x
Αν x mod 2 = 0 τότε
  y ← x div 2
  Αν y  ≤ 10 τότε
    y ← 2*x + y
  Τέλος_αν
αλλιώς
  y ← x^2
Τέλος_αν
Εμφάνισε y
Τέλος
```

Να ξαναγράψετε τον κώδικα ώστε να δίνει τα ίδια αποτελέσματα και ταυτόχρονα να είναι δυνατό να διαβαστεί από άνθρωπο. (από τις εξετάσεις 2006)

Απάντηση

```
Αλγόριθμος Θέμα2_2006
Διάβασε x

xΖυγός ← x mod 2 = 0
Αν xΖυγός και x ≤ 20 τότε    ! ζυγός, μικρότερος-ίσος του 20
  y ← x div 2 + 2*x
```

```
αλλιώς_αν xZυγός τότε        ! ζυγός, μεγαλύτερος του 20
  y ← x div 2
αλλιώς                       ! περιττός, αφού απέτυχαν οι προηγούμενες
  y ← x^2
Τέλος_αν
Εμφάνισε y
Τέλος
```

Σε ένα μπλοκ **Αν-Αλλιως_Αν** *ο διερμηνευτής δεν επιλέγει την περίπτωση που ταιριάζει καλύτερα! (όπως θα έκανε ίσως ένας άνθρωπος). Ο διερμηνευτής επιλέγει την πρώτη* ***Αλλιώς_Αν*** *που θα ικανοποιηθεί. Η σειρά των συνθηκών στις* **Αλλιώς_Αν** *είναι πολύ σημαντική.*

Ο διερμηνευτής δεν θα μπερδευτεί καθόλου στις διαδοχικές συνθήκες **xZυγός** και **x≤20 και xZυγός**. Θα τις ελέγξει με την σειρά εμφάνισης, συνεπώς η δεύτερη θα ικανοποιηθεί μόνο για ζυγούς μεγαλύτερους από 20. Το **αλλιώς** θα ικανοποιηθεί μόνο για τους περιττούς, αφού οι δύο πρώτες συνθήκες έχουν καλύψει όλους τους άρτιους (δείτε και άσκηση 33).
Εδώ χρησιμοποιούμε την λογική μεταβλητή **xZυγός**, χωρίς να είναι απαραίτητη. Κάνει όμως τον κώδικα καθαρότερο και λίγο ταχύτερο, με κόστος μια θέση μνήμης (δείτε και άσκηση 45).

Άσκηση 12 - πρωτοβάθμια εξίσωση

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ζητά τους συντελεστές α, β της εξίσωσης πρώτου βαθμού $\alpha x+\beta=0$ και να δίνει την λύση. Να γίνεται πλήρη διερεύνηση της εξίσωσης και για τις περιπτώσεις που αυτή είναι αδύνατη ή αόριστη.

Απάντηση

```
Αλγόριθμος ΠρωτοβάθμιαΕξίσωση
Εμφάνισε "Δώστε τον συντελεστή του x : "
Διάβασε α
Εμφάνισε "Δώστε τον σταθερό όρο : "
Διάβασε β

Αν α = 0 και β = 0 τότε
  Εμφάνισε "Αόριστη εξίσωση (όλα τα x είναι λύσεις)"
αλλιώς_αν α = 0 και β ≠ 0 τότε
  Εμφάνισε "Αδύνατη εξίσωση (δεν υπάρχει λύση)"
αλλιώς
  Εμφάνισε "Η λύση είναι : ", -β/α
Τέλος_αν
Τέλος
```

Άσκηση 13 - ελάχιστη & μέγιστη χρέωση

Με το Διατραπεζικό Σύστημα Συναλλαγών (ΔΙΑ.Σ.) μπορούμε να κάνουμε ανάληψη μετρητών από Αυτόματη Ταμειακή Μηχανή (ΑΤΜ) μιας τράπεζας, χρησιμοποιώντας κάρτα άλλης τράπεζας. Οι αναλήψεις αυτές όμως χρεώνονται, με το 1% του ποσού της ανάληψης, με τον περιορισμό η χρέωση να μην είναι μικρότερη από 1€ αλλά ούτε μεγαλύτερη από 3€, ανεξαρτήτως του ποσού της ανάληψης. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ζητά το ποσό της ανάληψης που θέλουμε να κάνουμε και να υπολογίζει την χρέωση.

Απάντηση

```
Αλγόριθμος ΔΙΑΣ
Εμφάνισε "Δώστε το ποσό ανάληψης : "
Διάβασε ποσό
```

```
χρέωση ← ποσό * 0.01

Αν χρέωση < 1 τότε
  χρέωση ← 1
αλλιώς_αν χρέωση > 3 τότε
  χρέωση ← 3
Τέλος_αν

Εμφάνισε "Η χρέωση είναι : ", χρέωση
Τέλος
```

Άσκηση 14 - εξτρά χρέωση πέραν του παγίου

Μια εταιρεία κινητής τηλεφωνίας εφαρμόζει την εξής πολιτική χρέωσης: πάγιο 20€ τον μήνα για δωρεάν χρόνο ομιλίας 600 λεπτά. Αν ο πελάτης ξεπεράσει το όριο των 600 λεπτών χρεώνεται για τον επιπλέον χρόνο προς 0.20€ ανά 5 λεπτά ομιλίας (η χρέωση γίνεται στην αρχή του πεντάλεπτου είτε το εξαντλήσει είτε όχι). Σε όλες τις χρεώσεις προστίθεται ΦΠΑ 23%. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει τον συνολικό χρόνο ομιλίας σε λεπτά (για έναν μήνα) και να εμφανίζει τη συνολική χρέωσή.

Απάντηση

```
Αλγόριθμος ΧρέωσηΟμιλίας
Εμφάνισε "Δώστε τον χρόνο ομιλίας (λεπτά) : "
Διάβασε χρόνος

χρέωση ← 20                                  ! το πάγιο

Αν χρόνος > 60 τότε
  χρόνος ← χρόνος - 600                      ! βγάλε τα δωρεάν λεπτά
  εξτρά ← χρόνος div 5                       ! ολόκληρα 5λεπτα
  ! αν έχει περισσέψει χρόνος, τότε ..
  ! ..υπάρχει μη ολοκληρωμένο πεντάλεπτο
  Αν χρόνος mod 5 ≠ 0 τότε εξτρά ← εξτρά + 1  ! μη ολοκληρωμένο 5λεπτο

  χρέωση ← χρέωση + εξτρά * 0.2
Τέλος_αν

χρέωση ← χρέωση* 1.23 ! το ΦΠΑ

Εμφάνισε "Η χρέωση είναι : ", χρέωση
Τέλος
```

Άσκηση 15 - ελάχιστη χρέωση

Αν κάποιος φορολογούμενος υποβάλλει τη δήλωσή του ηλεκτρονικά μέσω του Διαδικτύου, θα έχει έκπτωση 2.5% στον φόρο που του αναλογεί, με τον περιορισμό το ποσό της έκπτωσης να μην είναι μεγαλύτερο από 118€. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ζητά το ποσό του φόρου και να υπολογίζει τον φόρο που τελικά θα πληρώσει αν υποβάλλει τη δήλωσή του ηλεκτρονικά.

Απάντηση

```
Αλγόριθμος ΈκπτωσηΔιαδίκτυο
Εμφάνισε "Δώστε το ποσό του φόρου : "
Διάβασε φόρος

έκπτωση ← φόρος * 0.025
Αν έκπτωση > 118 τότε έκπτωση ← 118
```

```
φόρος ← φόρος - έκπτωση

Εμφάνισε "Ο πληρωτέος φόρος είναι : ", φόρος
Τέλος
```

Άσκηση 16 - έλεγχος εισόδου

Μια εταιρεία ενοικίασης αυτοκινήτων χρεώνει την πρώτη ημέρα ενοικίασης προς 50 € και κάθε επόμενη ημέρα μέχρι και την 10η προς 25 €. Όμως, αν ένα αυτοκίνητο νοικιαστεί για περισσότερες από 10 ημέρες, τότε θα χρεωθεί όλες τις ημέρες προς 30 € την ημέρα. Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ που να ρωτά τις ημέρες που ενοικιάσθηκε ένα αυτοκίνητο μέχρι ο χρήστης να εισάγει θετικό ακέραιο. Έπειτα να εμφανίζει στην οθόνη τη χρέωσή του.

Απάντηση

```
ΠΡΟΓΡΑΜΜΑ Ενοικίαση
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: μέρες
  ΠΡΑΓΜΑΤΙΚΕΣ: χρέωση
ΑΡΧΗ
  ΓΡΑΨΕ "Δώστε τις ημέρες ενοικίασης : "
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΔΙΑΒΑΣΕ μέρες
    ΑΝ μέρες <= 0 ΤΟΤΕ
      ΓΡΑΨΕ "Οι μέρες πρέπει να είναι θετικός ακέραιος : "
    ΤΕΛΟΣ_ΑΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ μέρες > 0

  ΑΝ μέρες <= 10 ΤΟΤΕ
    χρέωση <- 50 + (μέρες - 1)*25
  ΑΛΛΙΩΣ
    χρέωση <- 30*μέρες
  ΤΕΛΟΣ_ΑΝ

  ΓΡΑΨΕ "Η χρέωση είναι : ", χρέωση
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Άσκηση 17 - μήκη τριγώνου

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει τρεις αριθμούς α, β και γ και να βρίσκει αν μπορούν να αποτελούν μήκη των πλευρών ενός τριγώνου.

Απάντηση

```
Αλγόριθμος ΜήκηΤριγώνου
Δεδομένα // α, β, γ //

είναιΤρίγωνο ← α < β + γ και β < α + γ και γ < α + β

Εμφάνισε είναιΤρίγωνο
Τέλος
```

*Δεν χρειάζεται η εντολή **Αν** για να δώσετε τιμή σε λογική μεταβλητή. Η λογική μεταβλητή παίρνει τιμή με φυσικό τρόπο μέσω της εκχώρησης.*

Άσκηση 18 - αντιστροφή ψηφίων 2ψήφιου

Να διαβασθεί ένας ακέραιος αριθμός, να ελεγχθεί αν είναι διψήφιος ή όχι και αν ναι, να γίνει αντιστροφή των ψηφίων του. Για παράδειγμα αν είναι 83 να γίνει 38.

Απάντηση

```
Αλγόριθμος ΑντιστροφήΔιψήφιου
```

```
Δεδομένα // α //

Αν α < 100 και α > 9 τότε
  α ← (α mod 10)*10 + α div 10
Τέλος_αν

Εμφάνισε α
Τέλος
```

Σε έναν φυσικό αριθμό ο τελεστής `mod` $10^{ν}$ *μας δίνει τα ν τελευταία ψηφία (κόβει ότι υπάρχει πιο μπροστά) ενώ ο τελεστής* `div` $10^{ν}$ *κόβει τα ν τελευταία ψηφία και αφήνει μόνο τα μπροστά. Το* $10^{ν}$ *πρέπει να γραφεί μέσα στον κώδικα ως συγκεκριμένος αριθμός (10, 100, 1000, ...) επειδή η ύψωση σε δύναμη έχει ως αποτέλεσμα πραγματικό.*

Άσκηση 19 - ώρα για βάψιμο

Για να βαφεί ένα δωμάτιο απαιτείται 1 κουτί μπογιά ανά 3 τετραγωνικά μέτρα. Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ το οποίο να ρωτά το εμβαδόν του δωματίου που θα βαφεί και να εμφανίζει στην οθόνη πόσα κουτιά μπογιάς θα πρέπει να αγορασθούν.

Ο αριθμός κουτιών πρέπει να είναι ακέραιος.

Απάντηση

```
ΠΡΟΓΡΑΜΜΑ ΩραΓιαΒάψιμο
ΣΤΑΘΕΡΕΣ
  το1κουτί = 3     ! 3 τετραγωνικά / κουτί μπογιάς
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: τμ, κουτιά
ΑΡΧΗ
  ΓΡΑΨΕ "Πόσα τετραγωνικά μέτρα είναι ο τοίχοι;"
  ΔΙΑΒΑΣΕ τμ

  κουτιά <- τμ div το1κουτί
  ΑΝ τμ mod το1κουτί > 0 ΤΟΤΕ       ! αν δεν διαιρείται ακριβώς, ..
    κουτιά <- κουτιά + 1            ! ..τότε θέλουμε κι άλλο κουτί
  ΤΕΛΟΣ_ΑΝ
  ΓΡΑΨΕ "Πρέπει να αγοράσετε ", κουτιά, " κουτιά"
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Δομή Επανάληψης

Άσκηση 20 - φρουρός

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει συνεχώς αριθμούς μέχρι να διαβάσει τον αριθμό 0. Έπειτα να εμφανίζει στην οθόνη το άθροισμα τους.

Απάντηση

```
Αλγόριθμος Άθροισμα
Αθρ ← 0
Αρχή_επανάληψης
  Διάβασε α
  Αθρ ← Αθρ + α
Μέχρις_ότου α ≠ 0 επανάλαβε
Εμφάνισε "Το άθροισμα είναι: ", Αθρ
Τέλος
```

Το μηδέν θεωρείται φρουρός στην εισαγωγή δεδομένων. Γενικά φρουρό (sentinel value) λέμε κάποια τιμή που της δίνουμε διαφορετική σημασία από τις άλλες.

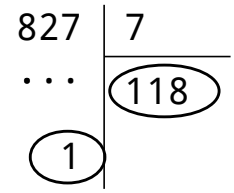
Άσκηση 21 - μικρότερο/μεγαλύτερο ως φίλτρο

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει συνεχώς αριθμούς μέχρι να διαβάσει τον αριθμό μηδέν. Στο τέλος να εμφανίζει στην οθόνη τον μικρότερο και μεγαλύτερο από τους αριθμούς που διάβασε. Το ίδιο το μηδέν να μην λαμβάνεται υπόψη στο μικρότερο και μεγαλύτερο. Λάβετε υπόψη την περίπτωση να δώσει ο χρήστης ως πρώτο αριθμό το μηδέν.

Απάντηση

```
Αλγόριθμος Μικρότερο_Μεγαλύτερο
Διάβασε α ! αφού διαβάσουμε το πρώτο στοιχείο
Αν α = 0 τότε
  Εμφάνισε "Δεν δόθηκε κανένας αριθμός"
αλλιώς ! ξεκινάμε τον αλγόριθμο
  μικρ ← α
  μεγ ← α
  Όσο α ≠ 0 επανάλαβε
    Αν α < μικρ τότε μικρ ← α
    Αν α > μεγ τότε μεγ ← α
    Διάβασε α
  Τέλος_επανάληψης
  Εμφάνισε "Το μικρότερο είναι: ", μικρ
  Εμφάνισε "Το μεγαλύτερο είναι: ", μεγ
Τέλος_αν
Τέλος
```

Άσκηση 22 - ευκλείδια διαίρεση με διαδοχικές αφαιρέσεις



Να υλοποιήσετε την ευκλείδια διαίρεση δύο φυσικών αριθμών σε μορφή διαδικασίας, χωρίς την χρήση των έτοιμων τελεστών **div** και **mod**. Συγκεκριμένα να γράψετε μια διαδικασία σε ΓΛΩΣΣΑ **Ευκλείδια(Δτος, δτης, π, υ)** , η οποία να δέχεται δύο ακέραιους αριθμούς **Δ** και **δ** και να επιστρέφει το πηλίκο **π** και το υπόλοιπο **υ** της διαίρεσης.

Απάντηση

```
ΔΙΑΔΙΚΑΣΙΑ Ευκλείδια(Δτος, δτης, π, υ)
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Δτος, δτης, π, υ
ΑΡΧΗ
  π ← 0                ! το αρχικό πηλίκο είναι μηδέν
  υ ← Δτος             ! το αρχικό υπόλοιπο είναι ολόκληρος ο Διαιρετέος
  ΟΣΟ υ > δτης ΕΠΑΝΑΛΑΒΕ ! Όσο χωράει κι άλλη φορά
    υ ← υ - δτης
    π ← π + 1          ! μέτρα πόσες φορές χώρεσε το δτης στο Δτος
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

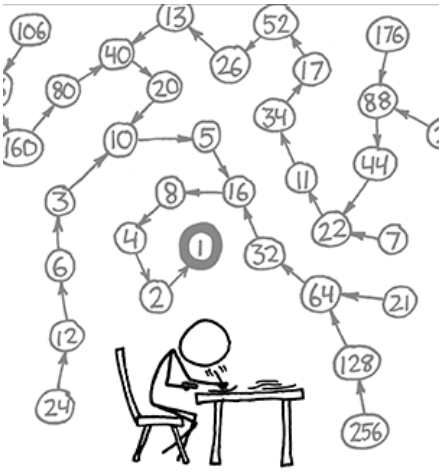
Άσκηση 23 - η εικασία του Collatz

Το παρακάτω πρόβλημα είναι γνωστό ως Εικασία του Collatz από το όνομα του γερμανού μαθηματικού που το πρότεινε. Για οποιοδήποτε φυσικό αριθμό εκτελούμε μια από τις δύο παρακάτω πράξεις.

(i) Αν είναι άρτιος τον διαιρούμε δια δύο.

(ii) Αν είναι περιττός τον πολλαπλασιάζουμε με 3 και προσθέτουμε 1. Σε μορφή τύπου:

$$f_{(n+1)}=\begin{cases} \dfrac{f_n}{2} & \alpha\nu\ f_n\ \acute{\alpha}\rho\tau\iota o \\ 3f_n+1 & \alpha\nu\ f_n\ \pi\varepsilon\rho\iota\tau\tau\acute{o} \end{cases}$$



Η εικασία υποστηρίζει ότι η ακολουθία καταλήγει πάντα στο 1. Για παράδειγμα αν ο αρχικός αριθμός είναι 7, έχουμε την ακολουθία (α) 7 είναι περιττό $3\cdot 7+1=22$, (β) 22 είναι άρτιο $22/2=11$, (γ) 11 είναι άρτιο $3\cdot 11+1=34$ ... και συνεχίζουμε μέχρι να φτάσουμε το 1.

Το πρόβλημα είναι ότι μέχρι τώρα δεν έχει βρεθεί κανείς αριθμός που να μην καταλήγει στο 1 αλλά ταυτόχρονα κανείς δεν μπόρεσε να αποδείξει ότι αυτό ισχύει για όλους τους αριθμούς. Μπορεί τελικά να υπάρχει τέτοιος αριθμός οπότε η εικασία του Collatz να αποδειχθεί λάθος.

Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ, το οποίο:

(α) Να ζητάει έναν φυσικό αριθμό από τον χρήστη μέχρι να δοθεί ένας έγκυρος φυσικός μεγαλύτερος από μηδέν.

(β) Να υπολογίζει τα βήματα της ακολουθίας Collatz γράφοντας το βήμα που εκτελεί και το αποτέλεσμα του κάθε βήματος.

(γ) Να σταματά όταν φτάσει στο 1. Αν δεν φτάσει ποτέ στο 1 τότε έχουμε βρει τον αριθμό που καταρρίπτει την εικασία! Είναι δυνατόν να καταρρίψουμε την εικασία με έναν τέτοιο αλγόριθμο;

(δ) Ένας τέτοιος αλγόριθμος παραβιάζει την αρχή της περατότητας; Μπορείτε να προτείνεται έναν τρόπο να μην παραβιάζεται η περατότητα;

Απάντηση

```
ΠΡΟΓΡΑΜΜΑ Collatz
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: φ, μετρ
ΑΡΧΗ
!(α) εισαγωγή φυσικού
  ΓΡΑΨΕ "Δώστε ένα φυσικό : "
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΔΙΑΒΑΣΕ φ
    ΑΝ φ <= 0 ΤΟΤΕ
      ΓΡΑΨΕ "Είπαμε φυσικός, πρόσεχε : "
    ΤΕΛΟΣ_ΑΝ
  ΜΕΧΡΙΣ_ΟΤΟΥ φ > 0

  ! Collatz
  μετρ <- 0
  ΟΣΟ φ > 1 ΕΠΑΝΑΛΑΒΕ ! (γ)
    ΑΝ φ mod 2 = 0 ΤΟΤΕ
      φ <- φ div 2
    ΑΛΛΙΩΣ
      φ <- φ* 3 + 1
    ΤΕΛΟΣ_ΑΝ
```

```
    μετρ <- μετρ + 1
    ΓΡΑΨΕ "Βήμα : ", μετρ, " -> ", φ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

(γ) Όχι. Δεν είναι δυνατόν να καταρρίψουμε την εικασία με τον αλγόριθμό μας. Το πρόβλημα είναι ότι όσα βήματα και αν κάνει ο αλγόριθμος ποτέ δεν θα είμαστε σίγουροι ότι δεν θα φτάσει στο 1 (ίσως μετά από μερικά ακόμη βήματα). Υπάρχει μια ειδική περίπτωση που μπορεί να καταρριφθεί η εικασία με έναν αλγόριθμο. Αν το αποτέλεσμα κάποιου βήματος είναι ίδιο με κάποιο προηγούμενο βήμα τότε δημιουργείται έναν ατέλειωτος βρόγχος και δεν φτάνουμε ποτέ στο 1. Θα πρέπει όμως να τροποποιήσουμε τον αλγόριθμο ώστε να θυμάται το αποτέλεσμα του κάθε βήματος και να συγκρίνει όλα τα επόμενα βήματα με τα προηγούμενα.

(δ) Η απάντηση εξαρτάται από το αν ισχύει ή όχι η εικασία. Αν η εικασία ισχύει, τότε όχι, δεν παραβιάζεται το κριτήριο της περατότητας. Αν όμως η εικασία δεν ισχύει, τότε ναι, παραβιάζεται το κριτήριο της περατότητας!

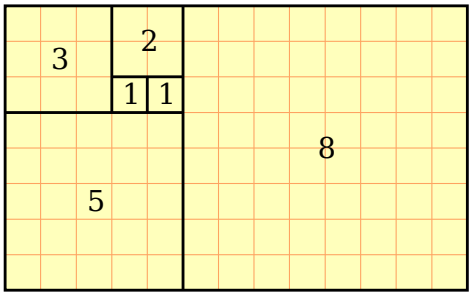
Για τον αλγόριθμό μας δεν υπάρχει πρόβλημα επειδή όλοι οι φυσικοί αριθμοί που μπορούν να αναπαρασταθούν από τους σημερινούς υπολογιστές (~10^{19}) έχουν δοκιμαστεί και ικανοποιούν την εικασία. Έτσι όποιο αριθμό και να εισάγουμε, ο αλγόριθμος θα τερματίσει. Αν όμως θέλαμε να δοκιμάσουμε αριθμούς που δεν έχουν δοκιμαστεί πριν, τότε πρέπει να τροποποιήσουμε τον αλγόριθμο ώστε να είμαστε σίγουροι ότι θα μπορεί να τερματίσει. Σε τέτοιες περιπτώσεις η πιο απλή μέθοδος είναι να δώσουμε στον χρήστη την δυνατότητα να διακόψει την εκτέλεση. Κάθε φορά που συμπληρώνεται ένας συγκεκριμένος αριθμός επαναλήψεων (π.χ. 1000) ρωτάμε τον χρήστη αν θέλει να συνεχίσουμε ή όχι. Ο κώδικας γίνεται κάπως έτσι

```
ΠΡΟΓΡΑΜΜΑ Collatz
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: φ, μετρ
  ΧΑΡΑΚΤΗΡΕΣ: απάντηση
  ΛΟΓΙΚΕΣ: συνέχισε
ΑΡΧΗ
...
  μετρ <- 0
  συνέχισε <- ΑΛΗΘΗΣ
  ΟΣΟ φ > 1 ΚΑΙ συνέχισε ΕΠΑΝΑΛΑΒΕ
...
    ΓΡΑΨΕ "Βήμα : ", μετρ, " -> ", φ
    ΑΝ μετρ mod 1000 = 0 ΤΟΤΕ                ! κάθε 1000 φορές
      ΓΡΑΨΕ "Θέλετε να συνεχίσετε? (ναι/όχι) "
      ΔΙΑΒΑΣΕ απάντηση
      συνέχισε <- απάντηση = 'ναι'
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Άσκηση 24 - ακολουθία Fibonacci

Η ακολουθία Fibonacci είναι μια αναδρομική ακολουθία ακέραιων αριθμών. Το κάθε νέο μέλος της ακολουθίας ορίζεται ως το άθροισμα των δύο προηγούμενων μελών. Αν δηλαδή δύο συνεχόμενα μέλη της ακολουθίας είναι $F_{(n-1)}$ και $F_{(n-2)}$ τότε ο επόμενος αριθμός της ακολουθίας είναι το άθροισμα $F_{(n-1)}+F_{(n-2)}$. Σε αναδρομικό τύπο: $F_n=F_{n-1}+F_{n-2}$

Για να αρχίσει η ακολουθία να δίνει αποτελέσματα πρέπει να ορίσουμε αυθαίρετα τους δύο πρώτους αριθμούς. Παραδοσιακά η επιλογή των δύο πρώτων όρων είναι $F_0=0$ και $F_1=1$. Οι πρώτοι αριθμοί της σειράς είναι 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...



Πλακόστρωση Fibonacci. Κάθε νέο πλακάκι έχει πλευρά ίση με τον αριθμό Fibonacci.

(α) Να γραφεί συνάρτηση σε ΓΛΩΣΣΑ η οποία να δέχεται ως παραμέτρους τον βαθμό του όρου που θέλουμε και να επιστρέφει την τιμή του αριθμού Fibonacci. Για παράδειγμα: **Φιμπο(11) = 89**, **Φιμπο(12) = 144**

(β) Οι μαθηματικοί έχουν βρει αναλυτικό τύπο για την ακολουθία Fibonacci, τον εξής: $F_n=\frac{(1+\sqrt{5})^n-(1-\sqrt{5})^n}{2^n\sqrt{5}}$. Θα ήταν προτιμότερο να γράψουμε την συνάρτηση των αριθμών Fibonacci στηριζόμενοι στον αναλυτικό τύπο ή με τον αναδρομικό τύπο του προηγούμενου ερωτήματος; Βλέπετε κάποιο μειονέκτημα στον αναλυτικό τύπο;

Απάντηση

(α)

```
ΣΥΝΑΡΤΗΣΗ Φιμπο(Ν): ΑΚΕΡΑΙΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Ν, α, β, κ
ΑΡΧΗ
  α <- 1                  ! ο τρέχων όρος (ο 2ος κατά την εκκίνηση)
  β <- 1                  ! ο προηγούμενος όρος (ο 1ος κατά την εκκίνηση)
  ΓΙΑ κ ΑΠΟ 3 ΜΕΧΡΙ Ν     ! ξεκινάμε από 3. Οι 2 πρώτοι όροι γνωστοί
    α <- α + β            ! νέο α = το άθροισμα των α και β
    β <- α - β            ! νέο β = το προηγούμενο α
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  Φιμπο <- α
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

(β) Με τον αναλυτικό τύπο ο υπολογισμός είναι αποδοτικότερος αφού δεν χρειαζόμαστε τις επαναλήψεις. Βρίσκουμε οποιοδήποτε όρο της ακολουθίας με λίγες μόνο πράξεις. Το μειονέκτημα είναι ότι το αποτέλεσμα δεν μπορεί να είναι ακέραιος αλλά πραγματικός, λόγω της ρίζας.

Άσκηση 25 - Μέγιστος Κοινός Διαιρέτης

Ο αλγόριθμος του Ευκλείδη για την εύρεση του μέγιστου κοινού διαιρέτη δύο φυσικών αριθμών διατυπώθηκε από τον Ευκλείδη το 300π.Χ. και παραμένει μέχρι σήμερα ο ταχύτερος αλγόριθμος που υπάρχει. Ο αλγόριθμος στηρίζεται σε δύο τύπους, γνωστούς από τον Πυθαγόρα.

(i) Αν ο ένας από του δύο αριθμούς είναι μηδέν, τότε ο ΜΚΔ είναι ο άλλος αριθμός, δηλαδή $\text{ΜΚΔ}(\alpha,0)=\alpha$

(ii) Αν από τους δύο αριθμούς ο α είναι μεγαλύτερος (ή ίσος) από τον β, τότε ο ΜΚΔ είναι ίσος με τον ΜΚΔ του μικρότερου και του υπόλοιπου της διαίρεσης του μεγαλύτερου με τον μικρότερο, δηλαδή $\text{ΜΚΔ}(\alpha,\beta)=\text{ΜΚΔ}(\beta,\alpha \bmod \beta)$

Με βάση τους τύπους, ο αλγόριθμος υπολογίζει επαναληπτικά τον ΜΚΔ του μικρότερου από τους δύο αριθμούς με το υπόλοιπο της διαίρεσης του μεγαλύτερου με τον μικρότερο, μέχρι ο δεύτερος αριθμός να γίνει μηδέν (το υπόλοιπο της διαίρεσης να γίνει μηδέν). Τότε ο ΜΚΔ είναι ο αριθμός που έμεινε (ο μεγαλύτερος εκείνη την στιγμή). Παράδειγμα με τους αριθμούς 18 και 48:
ΜΚΔ(48, 18) = ΜΚΔ(18,12) = ΜΚΔ(12, 6) = ΜΚΔ(6, 0) = 6
Χρησιμοποιώντας τους παραπάνω τύπους να γράψετε συνάρτηση σε ΓΛΩΣΣΑ η οποία να δέχεται δύο φυσικούς αριθμούς και να επιστρέφει τον ΜΚΔ. Οι αριθμοί μπορούν να δίνονται σε οποιαδήποτε σειρά, χωρίς να είναι απαραίτητο να δώσουμε τον μεγαλύτερο ως πρώτο όρισμα. Θα πρέπει όμως να είναι θετικοί ακέραιοι, συνεπώς δεν χρειάζεται έλεγχος δεδομένων μέσα στην συνάρτηση.
Απάντηση

```
ΣΥΝΑΡΤΗΣΗ ΜΚΔ(α, β): ΑΚΕΡΑΙΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: α, β, υπολ
ΑΡΧΗ
  ΑΝ α < β ΤΟΤΕ                ! αν κληθεί η συνάρτηση με τους α και β..
    ΚΑΛΕΣΕ Αντιμετάθεσε(α, β)  ! ..ανάποδα, τους αντιμεταθέτουμε
  ΤΕΛΟΣ_ΑΝ

  ! ο αλγόριθμος του Ευκλείδη
  ΟΣΟ β > 0 ΕΠΑΝΑΛΑΒΕ
    υπολ <- α mod β
    α <- β
    β <- υπολ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΜΚΔ <- α
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ

ΔΙΑΔΙΚΑΣΙΑ Αντιμετάθεσε(α, β)
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: α, β, προσ
ΑΡΧΗ
  προσ <- α
  α <- β
  β <- προσ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Άσκηση 26 - σχήμα Horner
Το σχήμα Horner είναι μια μέθοδος για να βρίσκουμε την τιμή ενός πολυωνύμου με τις λιγότερες δυνατές πράξεις. Παρουσιάστηκε το 1819 από τον Βρετανό μαθηματικό William Horner και παραμένει μέχρι σήμερα η ταχύτερη μέθοδος. Η μέθοδος χρησιμοποιήθηκε την δεκαετία του 70 στους πρώτους επεξεργαστές, για την υλοποίηση του πολλαπλασιασμού με χρήση μόνο πρόσθεσης.
Η μέθοδος απλά μετασχηματίζει το πολυώνυμο σε μια σειρά από προσθέσεις και πολλαπλασιασμούς, ώστε να αποφύγουμε την ύψωση σε δύναμη. Ο μετασχηματισμός δίνεται από τον παρακάτω τύπο

$$\sum_{\kappa=0}^{n} a_k x^k = a_0 + x(a_1 + x(a_2 + \ldots x(a_{n-1} + x a_n)))$$

Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ που να ζητά αρχικά την τιμή του x. Έπειτα να ζητά τον βαθμό του πολυωνύμου και μετά του συντελεστές έναν-έναν. Μόλις εισα-

χθεί ο τελευταίος συντελεστής θα πρέπει να εμφανίζει στην οθόνη την τιμή του πολυωνύμου.

Παρατηρήστε τον τύπο και βρείτε το μοτίβο της επανάληψης καθώς και την βολικότερη σειρά με την οποία θα ρωτάτε τους συντελεστές (από τον μεγιστοβάθμιο προς τον σταθερό όρο; ή το αντίστροφο;)

Απάντηση

Το μοτίβο της επανάληψης είναι η παρένθεση. Κάθε παρένθεση πολλαπλασιάζεται με χ και μετά προστίθεται ο συντελεστής. Αρχικά, στον n-στό όρο δεν υπάρχει παρένθεση, είναι δηλαδή μηδέν. Έτσι το αποτέλεσμα της κάθε επανάληψης πρέπει να είναι η παρένθεση, με αρχική τιμή μηδέν. Οι συντελεστές πρέπει να ζητούνται από τον μεγιστοβάθμιο προς τον σταθερό όρο.

```
ΠΡΟΓΡΑΜΜΑ Horner
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: ν, κ
  ΠΡΑΓΜΑΤΙΚΕΣ: πολυων, χ, α
ΑΡΧΗ
  ΓΡΑΨΕ "Δώστε τον βαθμό του πολυωνύμου : "
  ΔΙΑΒΑΣΕ ν
  ΓΡΑΨΕ "Δώστε την τιμή του χ : "
  ΔΙΑΒΑΣΕ χ

  πολυων <- 0                          ! η αρχική τιμή της παρένθεσης
  ΓΙΑ κ ΑΠΟ ν ΜΕΧΡΙ 0 ΜΕ_ΒΗΜΑ -1
    ΓΡΑΨΕ "Δώστε τον ", κ, "ο συντελεστή : "
    ΔΙΑΒΑΣΕ α
    πολυων <- πολυων*χ + α             ! η νέα παρένθεση
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΓΡΑΨΕ "Η τιμή του πολυωνύμου είναι : ", πολυων
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Άσκηση 27 - πλήθος ψηφίων (σωστή χρήση της ΜΕΧΡΙΣ_ΟΤΟΥ)

Να γραφεί συνάρτηση σε ΓΛΩΣΣΑ `ΠλήθοςΨηφίων(χ):ΑΚΕΡΑΙΑ` η οποία να δέχεται ως όρισμα έναν ακέραιο αριθμό και να επιστρέφει το πλήθος των ψηφίων του. Βεβαιωθείτε ότι δουλεύει σωστά για είσοδο μηδέν, αλλά και για αρνητικό αριθμό.

Απάντηση

Μπορούμε να μετρήσουμε τα ψηφία ενός φυσικού αριθμού (εκτός του μηδέν) με διαδοχικές ευκλείδιες διαιρέσεις με το 10. Κάθε φορά που εφαρμόζουμε **`div 10`** στον αριθμό, μειώνουμε τα ψηφία του κατά ένα. Αρκεί να μετρήσουμε πόσες διαιρέσεις θα χρειαστούν μέχρι να φτάσουμε στο μηδέν.

Το πρόβλημα είναι τι γίνεται αν μας δώσουν αρνητικό αριθμό ή αν μας δώσουν το μηδέν. Για να αντιμετωπίσουμε την περίπτωση του αρνητικού αριθμού, αρκεί να τον μετατρέψουμε σε θετικό με την ενσωματωμένη συνάρτηση της απόλυτης τιμής. Για να αντιμετωπίσουμε το πρόβλημα να μας δοθεί μηδέν, πρέπει να εφαρμόσουμε μία διαίρεση **`div 10`**, πριν αρχίσουμε τις επαναλήψεις. Έτσι καταλήγουμε στον παρακάτω κώδικα

```
ΣΥΝΑΡΤΗΣΗ ΠλήθοςΨηφίων(χ): ΑΚΕΡΑΙΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χ, πλήθος
ΑΡΧΗ
  χ <- Α_Τ(χ)              ! πιάνουμε την περίπτωση χ<0
  πλήθος <- 1              ! θέτουμε τον μετρητή ίσο με 1, για να..
```

```
  χ <- χ div 10           ! ..πιάσουμε την περίπτωση χ=0

  ! ο κυρίως αλγόριθμος
  ΟΣΟ χ <> 0 ΕΠΑΝΑΛΑΒΕ
    πλήθος <- πλήθος + 1
    χ <- χ div 10         ! σε κάθε επανάληψη ένα ψηφίο λιγότερο
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΠλήθοςΨηφίων <- πλήθος
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Ο προηγούμενος κώδικας είναι χαρακτηριστική περίπτωση λάθους χρήσης της **ΟΣΟ-ΕΠΑΝΑΛΑΒΕ**. Αφού απαιτούμε οπωσδήποτε μία εκτέλεση του σώματος της επανάληψης πριν αρχίσει η επανάληψη, θα πρέπει να χρησιμοποιήσουμε τη δομή **ΜΕΧΡΙΣ_ΌΤΟΥ**. Ο κώδικας γίνεται κομψότερος και καθαρότερος, χωρίς να χρειάζεται ιδιαίτερο χειρισμό για το μηδέν.

```
ΣΥΝΑΡΤΗΣΗ ΠλήθοςΨηφίων(χ): ΑΚΕΡΑΙΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χ, πλήθος
ΑΡΧΗ
  χ <- Α_Τ(χ)               ! πιάνουμε την περίπτωση χ<0
  πλήθος ← 0
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
      χ <- χ div 10          ! σε κάθε επανάληψη ένα ψηφίο λιγότερο
      πλήθος ←  πλήθος + 1
  ΜΕΧΡΙΣ_ΌΤΟΥ χ=0

  ΠλήθοςΨηφίων <- πλήθος
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Άσκηση 28 - άθροισμα ψηφίων

Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να ζητάει έναν φυσικό αριθμό και μετά να εκτυπώνει το άθροισμα των ψηφίων του. Για παράδειγμα για τον αριθμό 4356 το άθροισμα των ψηφίων του είναι 18.

Απάντηση

```
Αλγόριθμος ΆθροισμαΨηφίων
Διάβασε χ

αθρ ← 0
Όσο χ > 0 επανάλαβε
  ψηφίο ← χ mod 10       ! το τελευταίο ψηφίο του χ
  χ ← χ div 10           ! κόψε το τελευταίο ψηφίο του χ
  αθρ ← αθρ + ψηφίο
Τέλος_επανάληψης

Εμφάνισε αθρ
Τέλος
```

Άσκηση 29 - αντιστροφή ψηφίων

Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ το οποίο να ζητάει έναν φυσικό αριθμό και μετά να υπολογίζει έναν νέο αριθμό με τα αντίστροφα ψηφία του αρχικού. Για παράδειγμα ο αριθμός 31 θα γίνει 13, ο αριθμός 2378 θα γίνει 8732 και ο 78789 θα γίνει 98787. Ο νέος αριθμός να αποθηκεύεται σε μια θέση μνήμης και να εμφανίζεται στην οθόνη.

Απάντηση

Θα χρειαστούμε τη συνάρτηση **ΠλήθοςΨηφίων(χ)** από την άσκηση 27.

```
ΠΡΟΓΡΑΜΜΑ ΑντιστροφήΨηφίων
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χ, κ, πλΨηφ, ψηφίο
  ΠΡΑΓΜΑΤΙΚΕΣ: ψ
ΑΡΧΗ
  ΔΙΑΒΑΣΕ χ

  πλΨηφ <- ΠλήθοςΨηφίων(χ)
  ψ <- 0                    ! ψ είναι ο αντίστροφος αριθμός
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ πλΨηφ
    ψηφίο <- χ mod 10              ! κρατάμε το τελευταίο ψηφίο του χ..
    χ <- χ div 10                  ! ..και μετά το κόβουμε
    ψ <- ψ + ψηφίο* 10^(πλΨηφ - κ) ! η νέα αξία του ψηφίου
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ Α_Μ(ψ) ! θέλουμε το ψ σε ακέραιο
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Άσκηση 30 - άθροισμα/γινόμενο πολλών αριθμών

Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ το οποίο να διαβάζει 10 φυσικούς αριθμούς διάφορους από μηδέν και έπειτα να τυπώνει το άθροισμα όσων απ' αυτούς είναι ζυγοί και το γινόμενο όσων απ' αυτούς διαιρούνται με το 3.

Απάντηση

```
ΠΡΟΓΡΑΜΜΑ ΔέκαΑριθμοί
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χ, i, αθρ, γιν
ΑΡΧΗ
  αθρ <- 0                    ! όταν θέλουμε άθροισμα, ξεκινάμε από 0
  γιν <- 1                    ! όταν θέλουμε γινόμενο ξεκινάμε από 1
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    ! είσοδος από τον χρήστη
    ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
      ΔΙΑΒΑΣΕ χ
    ΜΕΧΡΙΣ_ΟΤΟΥ χ > 0

    ! επεξεργασία
    ΑΝ χ mod 2 = 0 ΤΟΤΕ
      αθρ <- αθρ + χ
    ΤΕΛΟΣ_ΑΝ
    ΑΝ χ mod 3 = 0 ΤΟΤΕ
      γιν <- γιν * χ
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ αθρ, γιν
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Άσκηση 31 - τετράγωνο ακεραίου μόνο με αφαίρεση

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να υπολογίζει το τετράγωνο ενός ακεραίου αριθμού χρησιμοποιώντας μόνο αφαίρεση. Δεν επιτρέπεται να χρησιμοποιήσουμε κανέναν τελεστή από τους `*`, `/`, `+`, `^` πουθενά στον αλγόριθμο. Μπορούμε όμως να χρησιμοποιήσουμε ελεύθερα αφαίρεση και τις ενσωματωμένες συναρτήσεις της ψευδογλώσσας.

Υπόδειξη: λύστε πρώτα το πρόβλημα χρησιμοποιώντας μόνο πρόσθεση και μετά θυμηθείτε ότι η πρόσθεση και η αφαίρεση είναι ουσιαστικά οι ίδιες πράξεις με διαφορετική φορά πάνω στην ευθεία των ακεραίων.

Απάντηση

Το τετράγωνο ενός αριθμού **Ν** είναι το γινόμενο του αριθμού με τον εαυτό του. Μπορούμε να το υπολογίσουμε με δύο φωλιασμένες επαναλήψεις. Η αφαίρεση μπορεί εύκολα να γίνει πρόσθεση αν μετράμε ανάποδα. Επειδή ο αριθμός μπορεί να είναι αρνητικός, χρησιμοποιούμε την απόλυτη τιμή του ως όριο στις επαναλήψεις.

```
τετράγωνο ← 0
Για κ από 1 μέχρι Α_Τ(Ν)
  Για λ από 1 μέχρι Α_Τ(Ν)
    τετράγωνο ← τετράγωνο - 1
  Τέλος_επανάληψης
Τέλος_επανάληψης

τετράγωνο ← Α_Τ(τετράγωνο)  ! το τετράγωνο είναι πάντα θετικό
```

Άσκηση 32 - παραγοντικό με επανάληψη

Να γραφεί συνάρτηση **Παραγ(ν):ΑΚΕΡΑΙΑ** σε ΓΛΩΣΣΑ η οποία να υπολογίζει το παραγοντικό ενός φυσικού αριθμού. Το παραγοντικό $n!$ ορίζεται από τον τύπο $n!=1\cdot 2\cdot 3\cdot \ldots \cdot n$. Κατά σύμβαση το παραγοντικό του μηδέν θεωρείται 1 ($0!=1$). Η συνάρτηση θα πρέπει να δέχεται έναν φυσικό αριθμό και να επιστρέφει το παραγοντικό του. Αν δεν δοθεί φυσικός τότε να επιστρέφει -1, ως ένδειξη λάθους. Παραδείγματα εκτέλεσης **Παράγ(6) = 720**, **Παράγ(10) = 3628800**, **Παράγ(13) = 6227020800**, **Παράγ(-3) = -1**

Προσοχή, μην δώσετε όρισμα παραπάνω από 20. Η τιμή του παραγοντικού ξεπερνά το όριο των ακεραίων που μπορεί να αναπαραστήσει ο υπολογιστής.

Απάντηση

```
ΣΥΝΑΡΤΗΣΗ Παραγ(ν): ΑΚΕΡΑΙΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: ν, γιν, κ
ΑΡΧΗ
  Παραγ <- -1        ! αν κάτι πάει στραβά, η συνάρτηση θα επιστρέψει -1
  ΑΝ ν >= 0 ΤΟΤΕ
    γιν <- 1
    ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ ν
      γιν <- γιν * κ
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    Παραγ <- γιν
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Άσκηση 33 - σωστή χρήση της "αλλιώς_αν"

Να διαβασθούν δύο αριθμοί α και β, όπου α < β, καθώς και 10 τυχαίοι άλλοι αριθμοί και να βρεθεί το πλήθος των αριθμών που είναι μικρότεροι του α, μεγαλύτεροι του β και μεταξύ α και β (περιλαμβανομένων των α και β).

Απάντηση

Όταν χρησιμοποιούμε **ΑΛΛΙΩΣ_ΑΝ** τότε η δομή της επιλογής σταματά μόλις ικανοποιηθεί η πρώτη συνθήκη. Έτσι η σειρά εμφάνισης των συνθηκών είναι πολύ σημαντική. Πολλές φορές παίζουμε με την σειρά των **ΑΛΛΙΩΣ_ΑΝ** ώστε να γλυτώσουμε συγκρίσεις.

Για να επιλεγεί μια **ΑΛΛΙΩΣ_ΑΝ** *θα πρέπει όχι μόνο να ικανοποιηθεί η συνθήκη της, αλλά και να αποτύχουν όλες οι προηγούμενες συνθήκες.*

```
ΠΡΟΓΡΑΜΜΑ ΑλλιώςΑν
ΜΕΤΑΒΛΗΤΕΣ
  ΠΡΑΓΜΑΤΙΚΕΣ: i, α, β, χ
  ΑΚΕΡΑΙΕΣ: μετρ1, μετρ2, μετρ3 ! οι μετρητές
ΑΡΧΗ
! αρχικοποίηση
  μετρ1 <- 0
  μετρ2 <- 0
  μετρ3 <- 0

! είσοδος δεδομένων
  ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
  ΔΙΑΒΑΣΕ α
  ΔΙΑΒΑΣΕ β
  ΜΕΧΡΙΣ_ΟΤΟΥ α < β

! ο κυρίως αλγόριθμος
  ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
  ΔΙΑΒΑΣΕ χ
  ΑΝ χ < α ΤΟΤΕ
    μετρ1 <- μετρ1 + 1
  ΑΛΛΙΩΣ_ΑΝ χ <= β ΤΟΤΕ ! ΚΑΙ χ>=α (αλλά δεν χρειάζεται να το ελέγξουμε)
    μετρ2 <- μετρ2 + 1
  ΑΛΛΙΩΣ ! ΑΝ χ>β (αλλά δεν χρειάζεται να το ελέγξουμε)
    μετρ3 <- μετρ3 + 1
  ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ μετρ1, μετρ2, μετρ3
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Άσκηση 34 - πρώτος αριθμός

Να γραφεί συνάρτηση **ΕίναιΠρώτος(χ):ΛΟΓΙΚΗ** σε ΓΛΩΣΣΑ, η οποία να δέχεται ως όρισμα **χ** έναν φυσικό αριθμό και να επιστρέφει **ΑΛΗΘΗΣ** αν ο αριθμός είναι πρώτος ή **ΨΕΥΔΗΣ** αν δεν είναι. Δεν χρειάζεται να κάνετε έλεγχο του ορίσματος, δηλαδή θα υποθέσετε ότι η συνάρτηση καλείται πάντα με θετικό ακέραιο, μεγαλύτερο από 1.

Απάντηση

Πρώτος είναι ένας φυσικός αριθμός αν διαιρείται ακριβώς μόνο με το 1 και τον εαυτό του. Ακριβώς πάνω στον ορισμό στηρίζεται ο αλγόριθμος. Θεωρούμε τον αριθμό πρώτο και δοκιμάζουμε διαδοχικά αν διαιρείται με τους φυσικούς από 2 και πάνω. Μπορούμε να επιταχύνουμε σημαντικά τον αλγόριθμο αν σταματήσουμε με τις δοκιμές μόλις φτάσουμε στην τετραγωνική ρίζα του **χ**. Αυτό συμβαίνει επειδή αν ο **χ** διαιρείται με έναν φυσικό **α**, τότε θα μπορεί να γραφεί ως γινόμενο **χ=αβ**. Τότε, ο ένας από τους δύο διαιρέτες θα είναι μικρότερος από την τετραγωνική ρίζα του **χ** και ο άλλος μεγαλύτερος ή και οι δύο θα είναι ίσοι με την τετραγωνική ρίζα του **χ** (αν ο **χ** είναι τέλειο τετράγωνο).

```
ΣΥΝΑΡΤΗΣΗ ΕίναιΠρώτος(χ): ΛΟΓΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χ, διαιρέτης
  ΛΟΓΙΚΕΣ: πρώτος
ΑΡΧΗ
  πρώτος <- ΑΛΗΘΗΣ                    ! Αρχικά θεωρείται πρώτος
  διαιρέτης <- 2                      ! ξεκινάμε τις διαιρέσεις από το 2
  ΟΣΟ πρώτος ΚΑΙ διαιρέτης < χ  ΕΠΑΝΑΛΑΒΕ
! ΟΣΟ πρώτος ΚΑΙ διαιρέτης <= Τ_Ρ(χ)  ΕΠΑΝΑΛΑΒΕ  ! βελτιωμένη έκδοση
    πρώτος <- ΟΧΙ χ mod διαιρέτης = 0
```

```
    διαιρέτης <- διαιρέτης + 1
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΕίναιΠρώτος <- πρώτος
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Άσκηση 35 - όλοι οι διαιρέτες (πρώτοι και σύνθετοι)

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ρωτάει από τον χρήστη έναν φυσικό αριθμό μεγαλύτερο του 1 και να εμφανίζει στην οθόνη όλους τους διαιρέτες του.

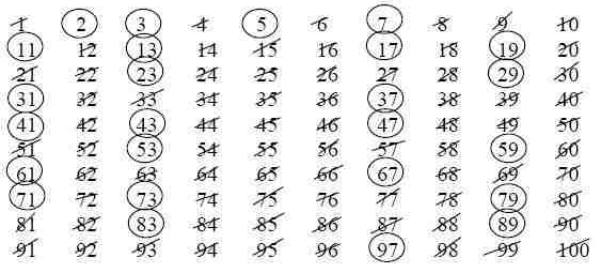
Απάντηση

Ο αλγόριθμος δοκιμάζει διαδοχικά αν ο **διαιρέτης** διαιρεί ακριβώς τον **χ**, για τιμές του **διαιρέτης** από 2 και πάνω. Έχουμε δύο επιλογές, (α) δοκιμάζουμε όλους του φυσικούς που είναι μικρότεροι από **χ** ή (β) όλους τους φυσικούς που είναι μικρότεροι ή ίσοι με την τετραγωνική ρίζα του **χ**. Η ιδέα είναι ότι όταν βρίσκουμε έναν διαιρέτη, τότε βρίσκουμε στην πραγματικότητα δύο διαιρέτες. Ο δεύτερος είναι το πηλίκο της διαίρεσης. Στην δεύτερη αυτή περίπτωση θα πρέπει να σταματήσουμε την επανάληψη όταν φτάσουμε σε τιμή ίση με την τετραγωνική ρίζα του **χ**, επειδή τότε θα έχουμε βρει όλους τους διαιρέτες.

```
Αλγόριθμος Διαιρέτες
Εμφάνισε "Δώστε έναν φυσικό αριθμό μεγαλύτερο του 1"
Διάβασε χ
Όσο χ < 2 και Α_Μ(χ) ≠ χ επανάλαβε
  Εμφάνισε "** Λάθος εισαγωγή"
  Εμφάνισε "Πρέπει ο αριθμός να είναι φυσικός, μεγαλύτερος του 1"
  Εμφάνισε "Δοκιμάστε ξανά..."
  Διάβασε  χ
Τέλος_επανάληψης
! Σε αυτό το σημείο έχουμε χ φυσικό μεγαλύτερο του 1

! κυρίως αλγόριθμος
διαιρέτης  ←  2
Όσο  διαιρέτης  <=  Τ_Ρ(χ)  επανάλαβε
  Αν χ mod διαιρέτης = 0 τότε
    Εμφάνισε διαιρέτης                ! ο ένας διαιρέτης
    Εμφάνισε χ div διαιρέτης          ! ο δεύτερος διαιρέτης
  Τέλος_αν
  διαιρέτης ← διαιρέτης + 1
Τέλος_επανάληψης
Τέλος
```

Ο προηγούμενος αλγόριθμος τυπώνει δύο φορές τον ίδιο διαιρέτη όταν ο χ είναι τέλειο τετράγωνο. Αυτό δεν είναι λάθος αλλά απλά ενοχλητικό. Ποιος είναι ο καλύτερος τρόπος να διορθώσουμε αυτήν την συμπεριφορά;

**** Άσκηση 36 - το κόσκινο του Ερατοσθένη***



Τον 3ο π.Χ. αιώνα ο Ερατοσθένης επινόησε ένα γρήγορο τρόπο να βρίσκουμε όλους τους πρώτους αριθμούς από το 2 μέχρι ένα μέγιστο. Αρχικά γράφουμε όλους τους αριθμούς σε πίνακα, κατά δεκάδες, εκατοντάδες ή και χιλιάδες. Αυτός ο πίνακας είναι το κόσκινο. Σβήνουμε πρώτα το 1 που δεν θεωρείται πρώτος και μετά επαναλαμβάνουμε επαναληπτικά:

ο μικρότερος αριθμός που δεν έχει σβηστεί είναι πρώτος. Τον κυκλώνουμε και σβήνουμε όλα τα πολλαπλάσιά του. Στο τέλος οι κυκλωμένοι αριθμοί είναι οι πρώτοι.
Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ που να υλοποιεί το κόσκινο του Ερατοσθένη. Το πρόγραμμα δεν χρειάζεται να ρωτά τίποτα από τον χρήστη. Θα δημιουργεί έναν πίνακα Ν θέσεων (έστω Ν=10000) και μετά θα τον γεμίζει με συνεχόμενους φυσικούς αριθμούς από το 1 μέχρι το Ν. Στη συνέχεια θα εφαρμόζει τον αλγόριθμο του Ερατοσθένη. Το σβήσιμο ισοδυναμεί με την τιμή 0 στον πίνακα. Στο τέλος ο πίνακας πρέπει να περιέχει μόνο τους πρώτους αριθμούς και 0 σε όσους αριθμούς δεν είναι πρώτοι. Ως έξοδο το πρόγραμμα πρέπει να εκτυπώνει όσα στοιχεία του πίνακα είναι διαφορετικά από 0 (το 0 χρησιμοποιείται ως φρουρός). Μήπως ο αλγόριθμός παραβιάζει το κριτήριο της εισόδου, αφού δεν ζητάει τίποτα από τον χρήστη;
** Το κόσκινο του Ερατοσθένη χρησιμοποιήθηκε για πολλούς αιώνες ως ο καλύτερος αλγόριθμος εύρεσης όλων των πρώτων αριθμών. Σήμερα θεωρείται ξεπερασμένο και πολύ απλό καθώς έχουν επινοηθεί πολλοί ταχύτερα κόσκινα. Παρόλα αυτά χρειάστηκαν 2000 χρόνια και ένας μεγάλος μαθηματικός μέχρι να έχουμε την πρώτη βελτίωση. Το 1719 ο Euler πρότεινε μια πολύ απλή μετατροπή που εκτοξεύει την ταχύτητα του αλγόριθμου. Συγκεκριμένα ο Euler παρατήρησε ότι ο αλγόριθμος σβήνει ξανά και ξανά τους ίδιους αριθμούς, ενώ με μια μικρή τροποποίηση μπορούμε να σβήνουμε τον κάθε αριθμό μία και μόνο φορά. Το αντίστοιχο κόσκινο λέγεται κόσκινο του Euler. Η τροποποίηση είναι τόσο απλή που είναι περίεργο πως δεν την σκέφτηκε ο ίδιος ο Ερατοσθένης αλλά και κανείς άλλος για 2000 χρόνια. Βρείτε αυτήν την βελτίωση του Euler παρατηρώντας τις περιττές διαγραφές των αριθμών και το αίτιο που τις προκαλεί. Μετά προτείνεται την κατάλληλη βελτίωση του αλγόριθμου. Σε ποιο σημείο θα έπρεπε να αλλάξει το πρόγραμμα που γράψατε σε ΓΛΩΣΣΑ;

Απάντηση

```
ΠΡΟΓΡΑΜΜΑ Ερατοσθένης
ΣΤΑΘΕΡΕΣ
  N = 100
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Α[Ν], κ, λ
ΑΡΧΗ
 ! Γέμισμα του πίνακα με τις αρχικές τιμές
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ Ν
    Α[κ] <- κ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

 ! Αφαιρούμε τον αριθμό 1 από τον πίνακα
  Α[1] <- 0      ! ότι είναι 0 στον πίνακα, σημαίνει ότι έχει σβηστεί

 ! Ο αλγόριθμος του Ερατοσθένη
  ΓΙΑ κ ΑΠΟ 2 ΜΕΧΡΙ Ν                  ! για κάθε στοιχείο του πίνακα
    ΑΝ Α[κ] <> 0 ΤΟΤΕ                   ! αν δεν έχει σβηστεί ήδη
      ! Αφού το κ δεν έχει σβηστεί άρα είναι πρώτος
      ! Θα πρέπει να σβήσουμε όλα τα πολλαπλάσια του κ...
      ! ..δηλαδή 2κ, 3κ,...(μέχρι το Ν div κ), έτσι ώστε...
      ! ..να μην βγούμε έξω από τον πίνακα
      ΓΙΑ λ ΑΠΟ 2 ΜΕΧΡΙ (Ν div κ)       ! ο πρωτότυπος αλγόριθμος
```

```
     !ΓΙΑ λ ΑΠΟ κ ΜΕΧΡΙ (N div κ)       ! η βελτίωση του Euler
        Α[κ*λ] <- 0                     ! σβήνουμε το πολλαπλάσιο του κ
      ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

 ! έξοδος αποτελεσμάτων
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ Ν
    ΑΝ Α[κ] <> 0 ΤΟΤΕ                   ! αν δεν έχει σβηστεί
      ΓΡΑΨΕ Α[κ]                        ! εμφάνισέ το
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Όχι, ο αλγόριθμος δεν παραβιάζει το κριτήριο της εισόδου. Είσοδος για τον αλγόριθμο είναι ο αριθμός Ν. Η είσοδος δεν είναι απαραίτητο να ζητείται από τον χρήστη. Μπορεί να είναι ενσωματωμένη στον κώδικα (hard-coded) ή να είναι ένα αρχείο στον δίσκο ή ένα κλικ του ποντικιού.

Άσκηση 37 - ανάλυση σε πρώτους παράγοντες

Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να ρωτάει από τον χρήστη ένα θετικό ακέραιο αριθμό και να τον αναλύει σε γινόμενο πρώτων παραγόντων. Για παράδειγμα για τον αριθμό 56 να τυπώνει διαδοχικά 2, 2, 2, 7 ενώ για τον αριθμό 57 να εκτυπώνει 3, 19.

Απάντηση

Ο αλγόριθμος είναι όμοιος με τον αλγόριθμο που χρησιμοποιούμε με χαρτί και μολύβι. Ξεκινώντας από το 2 δοκιμάζουμε όλους τους πρώτους αριθμούς αν διαιρούν τον αριθμό. Μόλις βρούμε έναν πρώτο παράγοντα, κάνουμε όσες διαιρέσεις μπορούν να γίνουν και μετά συνεχίζουμε τις δοκιμές με τον αμέσως μεγαλύτερο πρώτο αριθμό. Μετά από κάθε διαίρεση, ο αριθμός υποβαθμίζεται στο πηλίκο της διαίρεσης. Οι επόμενες δοκιμές γίνονται στο πηλίκο που έμεινε.

700	2
350	2
175	5
35	5
7	7
0	

Δυστυχώς δεν έχουμε την δυνατότητα να υλοποιήσουμε τον αλγόριθμο δοκιμάζοντας διαιρέσεις με πρώτους αριθμούς μόνο. Μπορούμε αντί γι' αυτό να δοκιμάζουμε όλους τους ακεραίους μεγαλύτερους του δύο. Η αποτελεσματικότητα του αλγορίθμου δεν επηρεάζεται επειδή, αφού βρούμε έναν πρώτο παράγοντα (πιθανόν πολλαπλές φορές) αποκλείεται να βρούμε ως “παράγοντα” κάποιο από τα πολλαπλάσια του (αφού έχουμε εξαντλήσει τις διαιρέσεις με τον πρώτο παράγοντα που βρήκαμε).

```
Αλγόριθμος ΠρώτοιΠαράγοντες
Διάβασε χ
ψ ← 2                       ! ο υποψήφιος πρώτος παράγοντας
Όσο χ > 1 επανάλαβε
  Αν χ mod ψ = 0 τότε
    Εμφάνισε ψ              ! έχουμε βρει έναν πρώτο παράγοντα
    χ ← χ div ψ
  αλλιώς
    ψ ← ψ + 1
  Τέλος_αν
Τέλος_επανάληψης
Τέλος
```

Πίνακες

Άσκηση 38 - άθροισμα διαγωνίου

Ένας δισδιάστατος τετραγωνικός πίνακας `A[N,N]` περιέχει πραγματικούς αριθμούς. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να προσθέτει τα στοιχεία της διαγωνίου.

Απάντηση

```
αθρ ← 0
Για κ από 1 μέχρι N
  αθρ ← αθρ + A[κ, κ]
Τέλος_επανάληψης
```

Άσκηση 39 - αναποδογύρισμα μονοδιάστατου πίνακα

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να “αναστρέφει” (αναποδογυρίζει) τα στοιχεία ενός μονοδιάστατου πίνακα `A[N]` θέσεων. Το πρώτο στοιχείο θα πρέπει να αντιμετατεθεί με το τελευταίο, το δεύτερο με το προτελευταίο, το τρίτο με το τρίτο από το τέλος κοκ.

Απάντηση

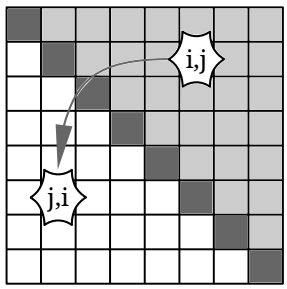
Αντιμεταθέτουμε το πρώτο με το τελευταίο, το δεύτερο με το προτελευταίο κτλ μέχρι να φτάσουμε στο μέσο του πίνακα. Αν το `A[κ]` σαρώνει τον πίνακα απ' την αρχή προς το τέλος, τότε το `A[N+1-κ]` τον σαρώνει από το τέλος προς την αρχή.

```
Για κ από 1 μέχρι N div 2
  Αντιμετάθεσε A[κ], A[N+1 - κ]
Τέλος_επανάληψης
```

Άσκηση 40 - αναστροφή τετραγωνικού πίνακα

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να αναστρέφει έναν τετραγωνικό πίνακα `A[N,N]`. Ανάστροφος είναι ο πίνακας που έχει τις γραμμές του πρώτου ως στήλες και τις στήλες ως γραμμές.

Απάντηση



Όσοι διδάχθηκαν πίνακες στα μαθηματικά, ξέρουν ότι για την αναστροφή αρκεί να αντιμεταθέσουμε, συμμετρικά ως προς την διαγώνιο, τα στοιχεία του πάνω τριγώνου. Πιο απλά ανταλλάσσουμε τον δείκτη της γραμμής με τον δείκτη της στήλης. Συνεπώς, τα στοιχεία της διαγωνίου θα παραμείνουν στις θέσεις τους. Οι υπόλοιποι, που δεν διδάχθηκαν πίνακες, μπορούν να το διαπιστώσουν εύκολα, κάνοντας ένα σχήμα στο χαρτί.

Για να πετύχουμε αυτή την αντιμετάθεση πρέπει να σαρώσουμε μόνο τα στοιχεία του πάνω τριγώνου. Το κόλπο είναι να ξεκινάμε τον δείκτη της εσωτερικής επανάληψης από θέση κατά ένα μεγαλύτερη απ' αυτή της εξωτερικής επανάληψης.

```
Για κ από 1 μέχρι N
  Για λ από κ+1 μέχρι N
    Αντιμετάθεσε A[κ,λ], A[λ,κ]
```

```
  Τέλος_επανάληψης
Τέλος_επανάληψης
```

Πολύ συχνά χρειάζεται να σαρώσουμε, είτε τετραγωνικό πίνακα, είτε έναν μονοδιάστατο πίνακα, με τέτοιο τρόπο ώστε να αποφύγουμε την σύγκριση των στοιχείων με τον εαυτό τους αλλά και τις διπλές συγκρίσεις μεταξύ των ζευγαριών (οι συγκρίσεις συνήθως είναι αντιμεταθετικές). Για να πετύχουμε αυτήν την ιδιαίτερη σάρωση εφαρμόζουμε την τεχνική αυτής της άσκησης. Η εξωτερική επανάληψη σαρώνει όλα τα στοιχεία, αλλά η εσωτερική σαρώνει τα στοιχεία από τον δείκτη της εξωτερικής συν ένα, μέχρι το τέλος. Αν δεν περιλάβουμε το συν ένα, τότε θα συγκρίνουμε και τα στοιχεία με τον εαυτό τους (ή τα στοιχεία της διαγωνίου, στην περίπτωση τετραγωνικού πίνακα)

Άσκηση 41 - ανάποδη αντιγραφή

Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να αντιγράφει τα στοιχεία ενός πίνακα **A[N]** σε ένα πίνακα **B[N]** αλλά με αντίστροφη σειρά.

Απάντηση

```
Για κ από 1 μέχρι N
  B[N+1 - κ] ← A[κ]
Τέλος_επανάληψης
```

Άσκηση 42 - πάνω από τον μέσο όρο

Ένας μονοδιάστατος πίνακας **A[N]** θέσεων περιέχει πραγματικούς αριθμούς. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να βρίσκει το επί τοις εκατό ποσοστό των στοιχείων του πίνακα που είναι μεγαλύτερα από τον μέσο όρο. Για παράδειγμα για τον πίνακα **[2.5, 3.2, 4.7, 1.1, 7.8]**, ο μέσος όρος είναι 3.86 και το ποσοστό των στοιχείων που είναι πάνω από το μέσο όρο είναι 40% (2 από τα 5 στοιχεία).

Απάντηση

Θα βρούμε πρώτα τον μέσο όρο και μετά θα μετρήσουμε πόσα στοιχεία είναι μεγαλύτερα.

```
! ο μέσος όρος
μο ← 0
Για κ από 1 μέχρι N
  μο ← μο + A[κ]
Τέλος_επανάληψης
μο ← μο / N

! μέτρημα των στοιχείων που είναι μεγαλύτερα από τον μέσο όρο
μετρητής  ← 0
Για κ από 1 μέχρι N
  Αν A[κ] > μο τότε
    μετρητής ← μετρητής + 1
  Τέλος_Αν
Τέλος_επανάληψης

! υπολογίζουμε το % ποσοστό
απάντηση ← 100 * μετρητής / N
```

Ένα μέρος μιας ποσότητας μετατρέπεται σε ποσοστό επί τοις εκατό διαιρώντας με τη συνολική ποσότητα και πολλαπλασιάζοντας επί 100.

Άσκηση 43 - καρκινικές φράσεις

Ένας μονοδιάστατος πίνακας `A[N]` περιέχει μία φράση, χαρακτήρα-χαρακτήρα (δηλαδή σε κάθε θέση του πίνακα είναι αποθηκευμένος ένας μόνο χαρακτήρας) με κεφαλαία γράμματα και χωρίς κενά. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ελέγχει αν η φράση του πίνακα είναι καρκινική ή όχι. Καρκινικές είναι οι φράσεις που μπορούν να διαβαστούν και από το τέλος προς την αρχή, για παράδειγμα ΑΝΝΑ, ΣΕΡΡΕΣ, ΝΙΨΟΝΑΝΟΜΗΜΑΤΑΜΗΜΟΝΑΝΟΨΙΝ.

Απάντηση

```
είναιΚαρκ ← αληθής
κ ← 1
Όσο είναιΚαρκ και κ <= N div 2
  Αν A[κ] ≠ A[N+1 - κ] τότε είναιΚαρκ ← ψευδής !ή είναιΚαρκ ← A[κ]=A[N+1-κ]
  κ ← κ + 1
Τέλος_επανάληψης
! εδώ το είναιΚαρκ δείχνει αν ο Α[] είναι καρκινικός ή όχι
```

Εδώ χρησιμοποιούμε μια λογική μεταβλητή ως σημαία (flag). Η σημαία είναι σαν ένας διακόπτης. Σε κάθε επανάληψη ελέγχουμε αν η σημαία είναι πάνω (on) ή κάτω (off) και ανάλογα αποφασίζουμε αν θα συνεχίσουμε την επανάληψη. Η σημαία παρακολουθεί κάποια συνθήκη, μέσα στην επανάληψη.

*** Το παρακάτω σχόλιο αφορά μόνο αυτούς που θέλουν να τα ξέρουν όλα. Τις περισσότερες φορές η σημαία μπορεί να παίρνει τιμή απ' ευθείας με εντολή εκχώρησης, χωρίς την χρήση* `Αν`*. Αυτό συμβαίνει και στην περίπτωση αυτής της άσκησης. Για τον αρχάριο ο κώδικας γίνεται λίγο πιο δύσκολος, γι' αυτό προτιμήστε να χρησιμοποιείται την αναλυτική μορφή της* `Αν` *(ή βάλτε σε σχόλιο και την εναλλακτική, αν θέλετε να δείξετε ότι τα ξέρετε όλα!). Η απλή εκχώρηση με την* `Αν` *δεν είναι πάντα ισοδύναμες. Η* `Αν` *θα αλλάξει την τιμή της σημαίας μόνο αν ικανοποιηθεί η συνθήκη ενώ η απλή εκχώρηση θα αλλάζει τιμή σε κάθε επανάληψη. Παραδοσιακά οι σημαίες σηματοδοτούν το τέλος της επανάληψης, συνεπώς το να αλλάζει τιμή κάθε φορά δεν αποτελεί πρόβλημα. Υπάρχουν όμως περιπτώσεις που αυτή η συμπεριφορά δεν είναι επιθυμητή. Δείτε στην άσκηση 88 μια τέτοια περίπτωση. Οι δύο τρόποι θα ήταν εντελώς ισοδύναμοι αν υπήρχε και* `αλλιώς` *το οποίο να δίνει την συμπληρωματική τιμή στην σημαία. Στην περίπτωση αυτή θα ήταν κουτό να χρησιμοποιηθεί* `Αν`*.*

Το ποια από τις δύο μορφές θα χρησιμοποιήσει κάποιος είναι θέμα περισσότερο προσωπικού στυλ (και αισθητικής). Στον πραγματικό κόσμο πάντως προτιμούμε, τις περισσότερες φορές, την απλή εκχώρηση από την `Αν` *(χωρίς να χρειάζεται να επεκταθούμε περισσότερο στο θέμα).*

Άσκηση 44 - αλυσιδωτή πρόσθεση

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να προσθέτει αλυσιδωτά δύο πίνακας `A[N]` και `B[M]` σε έναν τρίτο πίνακα `Γ[N·M]`. Θεωρείστε ότι οι πίνακες `A[]` και `B[]` έχουν δημιουργηθεί και περιέχουν δεδομένα. Επίσης ο `Γ[]` έχει την κατάλληλη διάσταση και έχει αρχικοποιηθεί. Στο τέλος του αλγορίθμου ο `Γ[]` θα πρέπει να περιέχει στις `N` πρώτες θέσεις τα στοιχεία του `A` και στις επόμενες `M` θέσεις τα στοιχεία του `B`. Για παράδειγμα

`A=[α,β,γ]` , `B=[γ,δ,ε]` τότε `Γ=[α,β,γ,γ,δ,ε]`

`Α=[α,β]` , `Β=[δ,α,β]` τότε `Γ=[α,β,δ,α,β]`

Απάντηση

```
! αντιγράφουμε πρώτα τον Α[]
Για κ από 1 μέχρι Ν
  Γ[κ] ← Α[κ]                  ! ένα προς ένα τα στοιχεία
Τέλος_επανάληψης

! αντιγράφουμε και τον Β
Για λ από 1 μέχρι Μ
  Γ[Ν + λ] ← Β[λ]              ! εδώ ξεκινάμε από τη θέση Ν του Γ
Τέλος_επανάληψης
```

Άσκηση 45 - κοντινότερο ζευγάρι μονοδιάστατου πίνακα

Ένας μονοδιάστατος πίνακας `Α[Ν]` περιέχει πραγματικούς αριθμούς. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να βρίσκει την ελάχιστη απόσταση μεταξύ δύο διπλανών στοιχείων. Απόσταση μεταξύ δύο στοιχείων είναι η απόλυτη τιμή της διαφορά τους. Για παράδειγμα στον πίνακα `[4,8,6,1,2,9,4]` η ελάχιστη απόσταση είναι 1, λόγω των στοιχείων `1` και `2` που βρίσκονται σε διπλανές θέσεις. Θεωρήστε ότι ο πίνακας έχει τουλάχιστον δύο στοιχεία.

Απάντηση

```
Αλγόριθμος ΚοντινότεροΖευγάρι
Ν ← 7
Α[1] ← 4
Α[2] ← 8
Α[3] ← 6
Α[4] ← 1
Α[5] ← 2
Α[6] ← 9
Α[7] ← 4

ελαχ ← Α_Τ(Α[1] - Α[2])   ! η πρώτη διαφορά του πίνακα είναι το ελάχιστο
Για κ από 2 μέχρι Ν-1     ! προσέξτε τα όρια της επανάληψης
  διαφ ← Α_Τ(Α[κ] - Α[κ+1])
  Αν διαφ < ελαχ τότε ελαχ ← διαφ
Τέλος_επανάληψης

Εμφάνισε ελαχ
Τέλος
```

Ο αλγόριθμος θα δούλευε το ίδιο καλά και χωρίς την επιπλέον μεταβλητή, δηλαδή

```
...
Για κ από 2 μέχρι Ν-1
  Αν Α_Τ(Α[κ] - Α[κ+1]) < ελαχ τότε
    ελαχ ← Α_Τ(Α[κ] – Α[κ+1])
  Τέλος_Άν
Τέλος_επανάληψης
...
```

Γλιτώνουμε έτσι μία θέση μνήμης, αλλά υποχρεώνουμε τον επεξεργαστή να κάνει δύο φορές τις ίδιες πράξεις, κάθε φορά που ικανοποιείται η συνθήκη. Έτσι ο πρώτος αλγόριθμος είναι ταχύτερος. Εδώ η διαφορά είναι ασήμαντη. Αν όμως χρησιμοποιούσαμε την μαθηματική παράσταση περισσότερες φορές, χωρίς να την αποδώσουμε πριν σε θέση μνήμης, τότε η διαφορά μπορεί να ήταν σημαντική.

Το μεγαλύτερο μειονέκτημα του δεύτερου κώδικα είναι ότι δεν είναι τόσο ευανάγνωστος όσο ο πρώτος. Γενικά προτιμήστε να θυσιάσετε λίγη μνήμη προκει-

μένου να κάνετε τον κώδικα πιο καθαρό. Να θυμάστε ότι η μνήμη γίνεται όλο και φτηνότερη, ενώ το κόστος του προγραμματιστή όλο και ακριβότερο.

Άσκηση 46 - ελάχιστη/μέγιστη διαφορά πίνακα

Ίδια με την προηγούμενη άσκηση, αλλά τώρα θέλουμε την ελάχιστη διαφορά ανάμεσα σε όλα τα στοιχεία και όχι μόνο μεταξύ των διπλανών. Για παράδειγμα στον πίνακα `[4,8,6,1,2,9,4]` της προηγούμενης άσκησης η ελάχιστη διαφορά είναι 0, λόγω των στοιχείων `4` και `4` που βρίσκονται στις ακραίες θέσεις.

Απάντηση

Ο μόνος τρόπος είναι με εξαντλητική σύγκριση όλων των δυνατών ζευγαριών. Μια πρώτη ιδέα θα ήταν φωλιασμένες επαναλήψεις, κάπως έτσι

```
...
Για κ από 1 μέχρι N
  Για λ από 1 μέχρι N
    ! (κ, λ) είναι όλα τα ζευγάρια
  Τέλος_επανάληψης
Τέλος_επανάληψης
```

Θα πρέπει βέβαια να αποκλείσουμε την σύγκριση του κάθε στοιχείου με τον εαυτό του, αλλά και τις διπλές συγκρίσεις των ίδιων ζευγαριών. Θα χρησιμοποιήσουμε την τεχνική που περιγράψαμε στην άσκηση 40, να ξεκινάμε τον δείκτη της εσωτερικής επανάληψης από θέση κατά ένα μεγαλύτερη απ' αυτή της εξωτερικής επανάληψης.

```
Αλγόριθμος ΕλάχιστηΔιαφορά
N ← 7
A[1] ← 4
A[2] ← 8
A[3] ← 6
A[4] ← 1
A[5] ← 2
A[6] ← 9
A[7] ← 4

ελαχ ← Α_Τ(A[1] - A[2])
Για κ από 1 μέχρι N - 1
  Για λ από κ + 1 μέχρι N
    διαφ ← Α_Τ(A[κ] - A[λ])
    Αν διαφ < ελαχ τότε ελαχ ← διαφ
  Τέλος_επανάληψης
Τέλος_επανάληψης

Εμφάνισε ελαχ
Τέλος
```

Άσκηση 47 - τοπικά μέγιστα (απομονωμένες κορυφές)

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να βρίσκει τα τοπικά μέγιστα ενός πίνακα `A[N]` πραγματικών αριθμών. Τοπικό μέγιστο είναι ένα στοιχείο του πίνακα το οποίο είναι μεγαλύτερο από τα γειτονικά του. Να θεωρήσετε ότι ο πίνακας δεν περιέχει δύο (ή περισσότερα) μέγιστα, ίσα μεταξύ τους και σε διπλανές θέσεις.

Απάντηση

Για τα άκρα του πίνακα κάνουμε χωριστό έλεγχο αφού εκεί έχουμε έναν μόνο γείτονα.

```
! ελέγχουμε το πρώτο στοιχείο
Αν A[1] > A[2] τότε
```

```
  Εμφάνισε Α[1], " στην θέση 1 είναι μέγιστο"
Τέλος_αν

! τα ενδιάμεσα στοιχεία
Για κ από 2 μέχρι Ν - 1
  Αν Α[κ] > Α[κ - 1] και Α[κ] > Α[κ + 1] τότε
    Εμφάνισε Α[κ], " στην θέση ", κ, " είναι μέγιστο"
  Τέλος_αν
Τέλος_επανάληψης

! ελέγχουμε το τελευταίο στοιχείο
Αν Α[Ν] > Α[Ν - 1] τότε
  Εμφάνισε Α[Ν], " στην θέση ", Ν, " είναι μέγιστο"
Τέλος_αν
```

Σημ. Ο αλγόριθμός μας θα δουλέψει στην περίπτωση που οι κορυφές είναι απομονωμένες. Δεν θα δουλέψει όμως στην περίπτωση ενός οροπεδίου ή μιας πλατιάς κορυφής.

Άσκηση 48 - πότε πρέπει να χρησιμοποιούμε πίνακα

Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ το οποίο να ζητάει τα ονόματα 20 μαθητών και τον βαθμό τους. Έπειτα να εμφανίζει στην οθόνη το όνομα του μαθητή με τον μεγαλύτερο βαθμό καθώς και την αρίθμηση που είχε αυτός κατά την εισαγωγή των δεδομένων (την σειρά με την οποία δώσαμε το όνομά του). Τι διαφορά θα έχει ο αλγόριθμος αν (α) ξέρουμε ότι όλοι οι βαθμοί είναι διαφορετικοί μεταξύ τους και (β) αν επιτρέπονται οι ισοβαθμίες; (στην δεύτερη αυτή περίπτωση θέλουμε όλα τα ονόματα και τις αριθμήσεις όλων των μαθητών που βγήκαν πρώτοι)

Απάντηση

(α) Αν ξέρουμε ότι όλοι οι βαθμοί είναι διαφορετικοί, τότε δεν χρειάζεται να θυμόμαστε ποιοι βαθμοί περάσαν. Αρκεί να θυμόμαστε ποιος βαθμός ήταν μεγαλύτερος (μαζί με το όνομα και την αρίθμηση). Αυτό σημαίνει ότι δεν χρειαζόμαστε πίνακα.

```
ΠΡΟΓΡΑΜΜΑ Μεγαλύτερος
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: κ, μεγΚ
  ΠΡΑΓΜΑΤΙΚΕΣ: βαθμός, μεγΒαθμός
  ΧΑΡΑΚΤΗΡΕΣ: όνομα, μεγΌνομα
ΑΡΧΗ
  μεγΒαθμός <- 0        ! αρχικά μεγαλύτερο θεωρούμε το μικρότερο δυνατό
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 20
    ΓΡΑΨΕ "Δώστε το όνομα: "
    ΔΙΑΒΑΣΕ όνομα
    ΓΡΑΨΕ "Δώστε τον βαθμός: "
    ΔΙΑΒΑΣΕ βαθμός
    ΑΝ βαθμός > μεγΒαθμός ΤΟΤΕ
      μεγΒαθμός <- βαθμός
      μεγΌνομα <- όνομα
      μεγΚ <- κ
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΓΡΑΨΕ "Μαθητής με τον μεγαλύτερο βαθμό: ", μεγΌνομα
  ΓΡΑΨΕ "με αρίθμηση: ", μεγΚ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

(β) Η κατάσταση είναι διαφορετική αν επιτρέπονται οι ίδιοι βαθμοί. Τότε μπορεί κάποιοι μαθητές να ισοβαθμίσουν και έτσι να προκύψουν πολλοί “μεγαλύτεροι” βαθμοί. Τώρα θα πρέπει να αποθηκεύουμε όλα τα ονόματα και τους βαθμούς σε πίνακα. Στο τέλος, όταν θα ξέρουμε ποιος είναι ο μεγαλύτερος βαθμός, θα σαρώσουμε τον πίνακα για να βρούμε ποιοι μαθητές τον πέτυχαν.

```
ΠΡΟΓΡΑΜΜΑ Μεγαλύτερος
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: κ, μεγΚ
  ΠΡΑΓΜΑΤΙΚΕΣ: Βαθ[20], μεγΒαθμός
  ΧΑΡΑΚΤΗΡΕΣ: Ον[20], μεγΌνομα
ΑΡΧΗ
  μεγΒαθμός <- 0 ! ο μεγαλύτερος βαθμός είναι το μικρότερο δυνατό
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 20
    ΓΡΑΨΕ "Δώστε το όνομα: "
    ΔΙΑΒΑΣΕ Ον[κ]
    ΓΡΑΨΕ "Δώστε τον βαθμός: "
    ΔΙΑΒΑΣΕ Βαθ[κ]
    ΑΝ Βαθ[κ] > μεγΒαθμός ΤΟΤΕ
      μεγΒαθμός <- Βαθ[κ]      !αρκεί μόνο να ξέρω τον μεγαλύτερο βαθμό
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

 ! εξαγωγή αποτελεσμάτων (σάρωση του πίνακα)
  ΓΡΑΨΕ "Μαθητής με τον μεγαλύτερο βαθμό (και η αρίθμησή του)"
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 20
    ΑΝ Βαθ[κ] = μεγΒαθμός ΤΟΤΕ
      ΓΡΑΨΕ Ον[κ], " ", κ
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

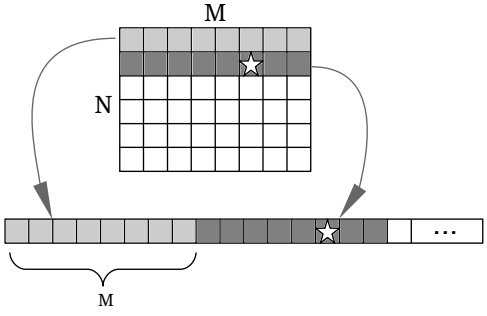
Οι αλγόριθμοι που μπορούν να δώσουν αποτέλεσμα χωρίς να θυμούνται ένα προς ένα τα δεδομένα που διάβασαν λέγονται φίλτρα ή στιγμιαίοι (instantaneous) ή on-line. Ο μέσος όρος και το μεγαλύτερο/μικρότερο στοιχείο είναι χαρακτηριστικά παραδείγματα φίλτρων. Ένα φίλτρο μπορεί να χρησιμοποιεί επιπλέον μνήμη, αρκεί το μέγεθος της μνήμης να είναι σταθερό και να μην εξαρτάται από το πλήθος των δεδομένων.
Υπάρχουν όμως αλγόριθμοι που πρέπει να θυμούνται όλα (ή σχεδόν όλα) τα δεδομένα που διάβασαν για να δώσουν αποτέλεσμα. Τότε λέμε ότι οι αλγόριθμοι χρειάζονται πίνακα (λέγονται επίσης μη-στιγμιαίοι ή off-line). Χαρακτηριστικό παράδειγμα off-line αλγορίθμου είναι η ταξινόμηση (εκτός από κάποιες πολύ τετριμμένες περιπτώσεις)

Άσκηση 49 - φίλτρα εναντίον πίνακα

Υποθέστε ότι ο χρήστης του υπολογιστή εισάγει μια σειρά από πραγματικούς αριθμούς. Δεν γνωρίζουμε εκ των προτέρων πόσοι θα είναι οι αριθμοί, αλλά μόνο ένα μέγιστο πλήθος που μπορούμε να δεχτούμε. Μόλις τελειώσει η εισαγωγή δεδομένων θέλουμε να εμφανίσουμε στην οθόνη τα παρακάτω αποτελέσματα:
(α) Τον μεγαλύτερο και μικρότερο αριθμό
(β) Τον 2ο και 3ο μεγαλύτερο αριθμό
(γ) Τον μεσαίο αριθμό με κριτήριο την σειρά εισαγωγής
(δ) Τον μεσαίο αριθμό με κριτήριο την ταξινόμησή τους

(ε) Το άθροισμα των τετραγώνων τους.
(στ) Την μέση τιμή τους (μέσος όρος)
(ζ) Το ποσοστό των αριθμών που ήταν μεγαλύτεροι από τον μέσο όρο.
(η) Όλους τους αριθμούς ταξινομημένους σε αύξουσα σειρά.
(θ) Όλους τους αριθμούς με αντίστροφη σειρά απ' αυτήν που διαβάστηκαν.
(ι) Πόσοι αριθμοί ήταν μεγαλύτεροι από 100 και πόσοι μικρότεροι
(κ) Ο αριθμός με την μεγαλύτερη συχνότητα εμφάνισης (υποθέστε ακεραίους αριθμούς μόνο αλλά με άπειρες δυνατές τιμές)
Σε ποιες από τις προηγούμενες περιπτώσεις θα χρειαστούμε πίνακα για την αποθήκευση των αριθμών και σε ποιες μπορούμε να γράψουμε τον αλγόριθμο ως φίλτρο;
Απάντηση
(α) Φίλτρο. Μετά από κάθε εισαγωγή, αρκεί να θυμόμαστε τον μικρότερο και μεγαλύτερο απ' όσους έχουν εισαχθεί μέχρι εκείνη τη στιγμή.
(β) Φίλτρο. Αρκεί να θυμόμαστε τους τρεις μεγαλύτερους μετά από κάθε εισαγωγή.
(γ) Πίνακας. Δεν χρειάζεται να θυμόμαστε όλους τους αριθμούς, αλλά τους μισούς πιο πρόσφατους. Προσέξτε ότι σε κάθε 2 καινούργιους αριθμούς που εισάγονται, μπορούμε να “ξεχνάμε” το πιο παλιό αριθμό. Το ποσό μνήμης που χρειαζόμαστε όμως εξαρτάται από το πλήθος των δεδομένων, οπότε δεν είναι φίλτρο.
(δ) Πίνακας. Η ταξινόμηση απαιτεί πλήρη γνώση όλων των δεδομένων. Είναι αδύνατο να ξέρουμε ποιος αριθμός θα είναι μεσαίος, πριν κάνουμε την ταξινόμηση.
(ε) Φίλτρο. Αρκεί να θυμόμαστε το άθροισμα των τετραγώνων των προηγούμενων αριθμών, χωρίς να ξέρουμε ποιοι ήταν οι αριθμοί.
(στ) Φίλτρο. Αρκεί να προσθέτουμε τους αριθμούς που διαβάζουμε και ταυτόχρονα να τους μετράμε.
(ζ) Πίνακας. Τον μέσο όρο μπορούμε να τον ξέρουμε μόνο μετά το τέλος της εισαγωγής. Για να βρούμε πόσοι αριθμοί ήταν μεγαλύτεροι από τον μέσο όρο πρέπει να σαρώσουμε όλα τα δεδομένα.
(η) Πίνακας. Χωρίς πίνακα η ταξινόμηση είναι δυνατή μόνο σε πολύ ειδικές περιπτώσεις (πχ ταξινόμηση με μέτρημα, εφόσον τα δεδομένα παίρνουν τιμές από περασμένο σύνολο)
(θ) Πίνακας.
(ι) Φίλτρο. Αρκεί να έχουμε δύο μετρητές και να τους ενημερώνουμε μετά από κάθε εισαγωγή.
(κ) Πίνακας. Εφόσον οι δυνατές τιμές των δεδομένων είναι άπειρες δεν μπορούμε να χρησιμοποιήσουμε την τεχνική του “μετρήματος” (δείτε και άσκηση 87)

Άσκηση 50 - δύο διαστάσεις σε μία



Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει έναν δισδιάστατο πίνακα `A[N,M]`, $N \times M$ θέσεων από ακεραίους. Μετά να μεταφέρει τα στοιχεία του Α σε έναν μονοδιάστατο πίνακα `B[N*M]`, πάλι $N \times M$ θέσεων αλλά σε μία διάσταση τώρα. Η μεταφορά να γίνεται γραμμή - γραμμή, δηλαδή να μεταφέρεται τη μία γραμμή

και αμέσως μετά την επόμενη. Τι αλλαγές πρέπει να κάνουμε στον κώδικα ώστε να μεταφέρονται τα στοιχεία στήλη προς στήλη;

Απάντηση

```
Αλγόριθμος ΔύοΔιαστάσειςΣεΜια
Εμφάνισε "Δώστε το πλήθος των γραμμών"
Διάβασε Ν
Εμφάνισε "Δώστε το πλήθος των στηλών"
Διάβασε Μ
! εισαγωγή πίνακα
Για κ από 1 μέχρι Ν
  Για λ από 1 μέχρι Μ
    Διάβασε Α[κ, λ]
  Τέλος_επανάληψης
Τέλος_επανάληψης

! μεταφορά σε μονοδιάστατο πίνακα
Για κ από 1 μέχρι Ν           ! τα στοιχεία της κ γραμμής πρέπει ..
  Για λ από 1 μέχρι Μ         ! ..να προσπεράσουν τα (κ-1)*Μ στοιχεία..
    Β[(κ - 1)*Μ + λ] ← Α[κ, λ] ! ..των προηγούμενων γραμμών και μετά ..
  Τέλος_επανάληψης            ! ..να μπουν στην λ θέση
Τέλος_επανάληψης

! εκτύπωσε τον νέο πίνακα
Για κ από 1 μέχρι Ν*Μ
  Εμφάνισε Β[κ], "  "
Τέλος_επανάληψης
Τέλος
```

Άσκηση 51 - κΦορές ο Α

Δύο πίνακες χαρακτήρων, `Α[Ν]` και `Β[κ·Ν]`, περιέχουν χαρακτήρες, αλλά έναν μόνο χαρακτήρα σε κάθε θέση. Γράψτε αλγόριθμο σε ψευδογλωσσα που να αντιγράφει τον πίνακα `Α`, κ φορές στον πίνακα `Β`. Για παράδειγμα αν `Α=[α,β]` και `κ=3`, τότε μετά την εκτέλεση του αλγορίθμου πρέπει `Β=[α,β,α,β,α,β]`, ενώ αν `Α=[χ,ψ,ω]` και κ=2 τότε πρέπει `Β=[χ,ψ,ω,χ,ψ,ω]`.

Απάντηση

```
Αλγόριθμος κΦορές
Ν ← 3
Α[1] ← 'χ'
Α[2] ← 'ψ'
Α[3] ← 'ω'
κ ← 2

δείκτης ← 1               ! ο δείκτης του νέου πίνακα
Για λ από 1 μέχρι κ       ! κ φορές η αντιγραφή
  Για μ από 1 μέχρι Ν     ! αντέγραψε τα Ν στοιχεία
    Β[δείκτης] ← Α[μ]
    δείκτης ← δείκτης + 1  ! ο δείκτης αυξάνεται συνεχώς, ασχέτως λ και μ
  Τέλος_επανάληψης
Τέλος_επανάληψης
Τέλος
```

Άσκηση 52 - σβούρα δεξιά/αριστερά

Ένας πίνακας χαρακτήρων `Α[Ν]` περιέχει χαρακτήρες, αλλά έναν μόνο χαρακτήρα σε κάθε θέση. Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να κάνει μια κυ-

κλική μετάθεση των στοιχείων προς τα δεξιά. Δηλαδή το πρώτο στοιχείο να γίνει δεύτερο, το δεύτερο τρίτο ...κτλ μέχρι το τελευταίο στοιχείο, το οποίο πρέπει να γίνει πρώτο. Για παράδειγμα αν `Α=[α,β,γ,δ]` τότε μετά την εκτέλεση του αλγορίθμου `Α=[δ,α,β,γ]`. Τι αλλαγές πρέπει να κάνετε στον αλγόριθμο για να λειτουργεί η μετάθεση αριστερόστροφα;

Απάντηση

Είναι μια γενίκευση της αντιμετάθεσης δύο αριθμών. Αρκεί να κρατήσουμε σε προσωρινή μνήμη το τελευταίο στοιχείο και μετά να μεταθέσουμε όλα τα υπόλοιπα. Τέλος βάζουμε την προσωρινή μνήμη στη σωστή θέση

```
Αλγόριθμος ΣβούραΔεξιά
Ν ← 4
Α[1] ← 'α'
Α[2] ← 'β'
Α[3] ← 'γ'
Α[4] ← 'δ'

προσ ← Α[Ν]                      ! κράτα το τελευταίο στοιχείο
Για κ από Ν μέχρι 2 με_βήμα -1   ! από το τέλος προς την αρχή
  Α[κ] ← Α[κ - 1]                ! το κάθε στοιχείο μία θέση δεξιά
Τέλος_επανάληψης
Α[1] ← προσ                      ! το τελευταίο στοιχείο γίνεται πρώτο

Για κ από 1 μέχρι Ν
  Εμφάνισε Α[κ], "  "
Τέλος_επανάληψης
Τέλος
```

Για να πετύχουμε την σβούρα προς τα αριστερά πρέπει να κρατήσουμε σε προσωρινή μνήμη το πρώτο στοιχείο, μετά να κάνουμε την μετάθεση των στοιχείων από την αρχή προς το τέλος (`Α[κ] ← Α[κ+1]` με όρια στην `Για-απο-μέχρι` το `1` έως `Ν-1`) και τέλος να τοποθετήσουμε την προσωρινή μνήμη στο τελευταίο στοιχείο.

Λογικές (και μαθηματικές) εκφράσεις

Άσκηση 53

Σε μια βάση δεδομένων έχουμε αποθηκευμένα όλα τα χωριά της Ελλάδας. Για κάθε χωριό έχουμε τρεις μεταβλητές:

```
ΛΟΓΙΚΗ: είναιΔήμος  ! δείχνει αν το χωριό είναι δήμος του “Καλλικράτης”
ΑΚΕΡΑΙΑ: πληθυσμός        ! το πλήθος των κατοίκων
ΠΡΑΓΜΑΤΙΚΗ: μέσοΜηνιάτικο ! ο μέσος μηνιαίος μισθός ανά κάτοικο
```

Θέλουμε να υπολογίσουμε μια νέα λογική μεταβλητή η οποία να δείχνει αν το χωριό είναι κεφαλοχώρι ή όχι. Κεφαλοχώρι θεωρείται ένα χωριό όταν συμβαίνει ένα από τα δύο (ή και τα δύο μαζί)

α) είναι δήμος και έχει περισσότερους από 5.000 κατοίκους

β) έχει περισσότερους από 10.000 κατοίκους και το συνολικό ετήσιο εισόδημα του χωριού είναι μεγαλύτερο από 100 εκατομμύρια €.

Να γραφεί μια γραμμή κώδικα η οποία να καταχωρεί την σωστή τιμή στην λογική μεταβλητή `είναιΚεφαλοχώρι`, χρησιμοποιώντας τις τρεις προηγούμενες μεταβλητές.

Απάντηση

```
είναιΚεφαλοχώρι ← (είναιΔήμος ΚΑΙ πληθυσμός > 5000) Ή ...
            ... (πληθυσμός > 10000 ΚΑΙ 12*πληθυσμός*μέσοΜηνιάτικο > 10^8)
```

Άσκηση 54 - στρογγυλοποίηση στο δεύτερο δεκαδικό

Σε ένα λογιστικό φύλλο εργασίας (πχ Excel), όταν αλλάξουμε την μορφή ενός κελιού σε νομισματική, τότε αυτομάτως ο αριθμός που περιέχει το κελί στρογγυλοποιείται στο δεύτερο δεκαδικό ψηφίο (επειδή εκφράζει ευρώ). Ο προγραμματιστής που έγραψε τον κώδικα καυχιέται ότι πέτυχε αυτήν την μετατροπή σε μία γραμμή κώδικα χωρίς να χρησιμοποιήσει επιπλέον μνήμη. Γράψτε μία γραμμή κώδικα η οποία να καταχωρεί έναν πραγματικό αριθμό α στην ίδια θέση μνήμης, αλλά στρογγυλεμένο στο δεύτερο δεκαδικό ψηφίο. Για παράδειγμα ο αριθμός 32,451 θα γίνει 32,45, ο αριθμός 2312,017 θα γίνει 2312,02 και ο αριθμός 0,996 θα γίνει 1,00.

Απάντηση

Τα βήματα για να πετύχουμε το ζητούμενο είναι:

(i) μετακινούμε την υποδιαστολή 2 θέσεις δεξιά (πολλαπλασιασμός με 100)
(ii) προσθέτουμε 0,5 για την στρογγυλοποίηση
(iii) κόβουμε το δεκαδικό μέρος που έμεινε (με την συνάρτηση `A_M()`)
(iv) ξαναγυρνάμε την υποδιαστολή στην παλιά της θέση (διαίρεση με 100)

Μπορούν πολύ εύκολα όλα τα βήματα να γίνουν σε μία γραμμή κώδικα χωρίς να χρειαστούμε επιπλέον θέση μνήμης

```
α ← A_M( (α*100) + 0.5 ) / 100
```

Άσκηση 55 - τρεις αριθμοί ίσοι μεταξύ τους

Έχουμε τρεις ακέραιους αριθμούς `α`, `β`, `γ`. Θέλουμε με μια λογική έκφραση για να ελέγξουμε ότι και οι τρεις αριθμοί είναι ίσοι, δηλαδή `α=β=γ`. Ποια λογική έκφραση θα γράψουμε;

Απάντηση

```
όλοιΊσοι ← α = β ΚΑΙ β = γ
```

Άσκηση 56 - και τα δύο ίδια

Δύο πίθηκοι ζουν μαζί σε ένα κλουβί ζωολογικού κήπου. Ο κάθε πίθηκος μπορεί να είναι χαρούμενος ή λυπημένος. Μια λογική μεταβλητή για τον καθένα `αΧαρούμενος` και `βΧαρούμενος` καθορίζει την ψυχική διάθεση του κάθε πιθήκου. Οι πίθηκοι ζουν αρμονικά αν είναι και οι δύο χαρούμενοι ή και οι δύο λυπημένοι, αλλιώς πρέπει να μπουν σε διαφορετικά κλουβιά. Μια λογική μεταβλητή `είναιΑρμονικοί` πρέπει να πάρει την τιμή `Αληθής` αν έχουν την ίδια ψυχική διάθεση, αλλιώς πρέπει να είναι `Ψευδής`. Να γραφεί μια γραμμή κώδικα που να δίνει την κατάλληλη τιμή στην μεταβλητή `είναιΑρμονικοί`.

Απάντηση

```
είναιΑρμονικοί ← (αΧαρούμενος ΚΑΙ βΧαρούμενος) Ή ..
                                ..(ΟΧΙ αΧαρούμενος ΚΑΙ ΟΧΙ βΧαρούμενος)
```

Παρατήρηση 1: Οι παρενθέσεις είναι περιττές επειδή ο τελεστής `ΚΑΙ` έχει μεγαλύτερη προτεραιότητα από το `Ή` (ο τελεστής `ΚΑΙ` είναι ο πολλαπλασιασμός για τις λογικές εκφράσεις και το `Ή` η πρόσθεση). Είναι σωστό όμως να βάζουμε τις παρενθέσεις για να είναι πιο κατανοητή η έκφραση.

Παρατήρηση 2: Αν και δεν είναι προφανές, η τελευταία έκφραση γράφεται επίσης

```
είναιΑρμονικοί ← (αΧαρούμενος ΚΑΙ βΧαρούμενος) ..
```

```
                                        ..Ή ΟΧΙ (αΧαρούμενος Ή βΧαρούμενος)
```

και έτσι γλιτώνουμε έναν τελεστή. Υπάρχει ένας απλός κανόνας για τον τελεστή `ΟΧΙ`. Όταν το `ΟΧΙ` βγαίνει κοινός παράγοντας από μια παρένθεση (ή μπαίνει μέσα σε μια παρένθεση), τότε μετατρέπει τα `ΚΑΙ` σε `Ή` και τα `Ή` σε `ΚΑΙ`. Είναι εκτός ύλης αλλά χρήσιμο.

Άσκηση 57 - το λιοντάρι που βρυχάται

Στον ίδιο ζωολογικό κήπο υπάρχει ένα λιοντάρι που βρυχάται πολύ δυνατά. Μια λογική μεταβλητή `βρυχάται`, καθορίζει αν το λιοντάρι βρυχάται κάποια χρονική στιγμή. Μια ακέραια μεταβλητή `ώρα` δείχνει την ώρα της ημέρας σε 24ώρη μορφή, δηλαδή παίρνει τιμές από 0 έως 23. Το λιοντάρι δεν πρέπει να βρυχάται πριν τις 8 και μετά τις 21, επειδή αναστατώνει τα υπόλοιπα ζώα που κοιμούνται. Να γραφεί μια γραμμή κώδικα που να δίνει την κατάλληλη τιμή στην λογική μεταβλητή `λιοντάριΟΚ`, χρησιμοποιώντας τις μεταβλητές `βρυχάται` και `ώρα`.

Απάντηση

Όταν δεν βρυχάται τότε δεν μας ενδιαφέρει τι ώρα. Όταν η ώρα είναι μεταξύ 8 και 21, τότε δεν μας ενδιαφέρει τι κάνει το λιοντάρι.

```
λιοντάριΟΚ ← ΟΧΙ βρυχάται Ή (ώρα >= 8 ΚΑΙ ώρα <= 21)
```

Παρατήρηση 1: Η παρένθεση είναι πάλι περιττή, αλλά καλό είναι να υπάρχει.

Παρατήρηση 2: Αν γράψατε κάτι πολύ πιο πολύπλοκο, τότε μπορεί να είναι σωστό. Ελέγξτε ότι δίνει το σωστό αποτέλεσμα με όλους τους συνδυασμούς `ώρα` και `βρυχάται`.

Παρατήρηση 3: Αν δεν τα πάτε καλά με τα προβλήματα αυτού του είδους τότε μη στεναχωριέστε, δεν είστε οι μόνοι. Ακόμη και οι πολύ έμπειροι προγραμματιστές τρέμουν τις λογικές εκφράσεις. Υπάρχει ολόκληρη θεωρία με κανόνες και κολπάκια για το πως απλοποιούνται. Για την ΑΕΠΠ δεν χρειάζονται όλα αυτά. Γράψτε την λογική έκφραση, όσο πολύπλοκη κι αν είναι, και μετά προσπαθήστε με απλή λογική να δείτε αν έχει κάτι περιττό. Όπως κι αν γραφεί, αν είναι σωστή, βαθμολογείται με άριστα.

Άσκηση 58 - ή το ένα ή το άλλο

Στον ίδιο ζωολογικό κήπο υπάρχουν δύο παπαγάλοι που έχουν μάθει και απαγγέλλουν ένα ποίημα. Δύο λογικές μεταβλητές `αΑπαγγέλει` και `βΑπαγγέλει` παίρνουν την κατάλληλη τιμή κάθε φορά που ένας παπαγάλος απαγγέλει. Το πρόβλημα είναι ότι όταν απαγγέλλουν και οι δύο μαζί, οι θεατές δεν καταλαβαίνουν το ποίημα. Να γραφεί μια λογική έκφραση που να καταχωρεί στην λογική μεταβλητή `παπαγάλοιΟΚ`, την τιμή `Αληθής` όταν απαγγέλει ο ένας από τους δύο παπαγάλους, αλλιώς να καταχωρεί `Ψευδής`.

Απάντηση

Είναι η κλασική περίπτωση του αποκλειστικού-ή, δηλαδή ή το ένα ή το άλλο, άλλα όχι και τα δύο μαζί.

```
ΠαπγάλοιΟΚ ← (αΑπαγγέλει ΚΑΙ ΟΧΙ βΑπαγγέλει) Ή ..
                                    ..(ΟΧΙ αΑπαγγέλει ΚΑΙ βΑπαγγέλει)
```

Οι παρενθέσεις είναι και πάλι περιττές για τον διερμηνευτή, αλλά πρέπει να υπάρχουν για τον προγραμματιστή.

Άσκηση 59 - έλεγχος για τέλειο τετράγωνο

Με ποια λογική έκφραση μπορώ να ελέγξω αν ένας ακέραιος αριθμός είναι τέλειο τετράγωνο; Για παράδειγμα για τους αριθμούς 4, 9, 16, 25, 36, … η λογική έκφραση πρέπει να παίρνει την τιμή `Αληθής`.

Απάντηση

Θα ελέγξουμε αν η τετραγωνική ρίζα του αριθμού είναι ακέραιος αριθμός

```
τέλειοΤετράγωνο ← T_P(χ) = A_M(T_P(χ))
```

Άσκηση 60 - έλεγχος ημερομηνίας (ο πίνακας ως βοηθητικό στοιχείο)

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ελέγχει αν μια ημερομηνία είναι σωστή. Η ημερομηνία δίνεται σε μορφή πίνακα ακεραίων `HM[3]` με τρεις θέσεις. Η πρώτη θέση είναι η ημέρα του μήνα, η δεύτερη θέση ο μήνας και η τρίτη θέση το έτος. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ελέγχει αν ο πίνακας εκφράζει σωστή ημερομηνία. Σωστές ημερομηνίες να θεωρούνται αυτές που το έτος είναι μεγαλύτερο-ίσο με 1901 και μικρότερο-ίσο από 2099. Ιδιαίτερη προσοχή χρειάζεται στον μήνα Φεβρουάριο όταν το έτος είναι δίσεκτο. Ειδικά για την περίπτωση που το έτος είναι μεταξύ 1901 και 2099, δίσεκτο έτος είναι αυτό που διαιρείται ακριβώς με το 4 (δεν είναι γενικός κανόνας).

Απάντηση

Πριν ξεκινήσετε να γράφεται ένα τεράστιο `Αν` (το οποίο θα βγει έξω από την οθόνη) δείτε έναν κώδικα που δεν χρησιμοποιεί καθόλου `Αν`.

```
ημρ ← HM[1]
μήν ← HM[2]
έτος ← HM[3]

ΗμΜήν[1] ← 31  !Ιανουάριος
ΗμΜήν[2] ← 28  !Φεβρουάριος
ΗμΜήν[3] ← 31  !Μάρτιος
ΗμΜήν[4] ← 30  !κτλ
ΗμΜήν[5] ← 31
ΗμΜήν[6] ← 30
ΗμΜήν[7] ← 31
ΗμΜήν[8] ← 31
ΗμΜήν[9] ← 30
ΗμΜήν[10] ← 31
ΗμΜήν[11] ← 30
ΗμΜήν[12] ← 31

σωστό ← έτος >1900 και έτος < 2100 και μήν > 0 και μήν < 13 και ημρ > 0
σωστό ← σωστό και (ημρ<=ΗμΜήν[μήν] ή (μην=2 και ημρ=29 και έτος div 4=0))

Εμφάνισε σωστό
```

Διάφορες ασκήσεις

Άσκηση 61 - ρέστα (ο πίνακας ως βοηθητικό στοιχείο)

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει ένα ποσό σε €, χωρίς δεκαδικά ψηφία και να το αναλύει σε χαρτονομίσματα και κέρματα, έτσι ώστε ο συνολικός αριθμός τους να είναι ο ελάχιστος δυνατός. Ακριβώς όπως υπολογίζουμε

τα ρέστα. Φανταστείτε ότι ο αλγόριθμος θα εκτελείτε στο γκισέ ενός σούπερ μάρκετ για να διευκολύνει τον ταμία στα ρέστα που επιστρέφει στον πελάτη. Τα δυνατά χαρτονομίσματα και κέρματα είναι 500, 200, 100, 50, 20, 10, 5, 2, 1. Παραδείγματα εκτέλεσης:
αν δώσουμε 105 μας δίνει 1 X 100 και 1 X 5
αν δώσουμε 524 μας δίνει 1 X 500 και 1 X 20 και 2 X 2
αν δώσουμε 887 μας δίνει 1 X 500 και 1 X 200 και 1 X 100 και 1 X 50 και 1 X 20 και 1 X 10 και 1 X 5 και 1 X 2

Υπόδειξη : χρησιμοποιήστε κατάλληλα τους τελεστές div και mod

Απάντηση

```
Αλγόριθμος Ρέστα

Εκτύπωσε "Ποιο είναι το ποσό σε ευρώ (χωρίς δεκαδικά)"
Διάβασε ποσό

! ορισμός νομισμάτων
νόμισμα[1] ← 500
νόμισμα[2] ← 200
νόμισμα[3] ← 100
νόμισμα[4] ← 50
νόμισμα[5] ← 20
νόμισμα[6] ← 10
νόμισμα[7] ← 5
νόμισμα[8] ← 2
νόμισμα[9] ← 1

Για i από 1 μεχρι 9
  ! τα ρέστα είναι πόσες φορές χωράει το νόμισμα στο ποσό
  ρέστα ←  ποσό div νόμισμα[i]
  ! αν τα ρέστα είναι 0 δεν χρειάζεται να κάνουμε τίποτα
  Αν ρέστα > 0 τότε
    Εκτύπωσε ρέστα, " X ", νόμισμα[i]
    ποσό ←  ποσό mod νόμισμα[i] ! ότι έμεινε για την επόμενη επανάληψη
  Τέλος_αν
Τέλος_επανάληψης

Τέλος
```

Λίγο αποδοτικότερο κώδικα μπορούμε να πετύχουμε αν σταματάμε την επανάληψη όταν το ποσό φτάσει στο μηδέν. Το σώμα της επανάληψης θα μπορούσε να γραφεί ως εξής:

```
i ← 0
Όσο i < 10 και ποσό > 0 επανάλαβε
  i ← i + 1
  ρέστα ←  ποσό div νόμισμα[i]
  Αν ρέστα > 0 τότε
    Εκτύπωσε ρέστα, " X ", νόμισμα[i]
    ποσό ←  ποσό mod νόμισμα[i]
  Τέλος_αν
Τέλος_επανάληψης
```

Άσκηση 62 - μην κάνετε τα πράγματα πιο δύσκολα απ' ότι είναι

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να τυπώνει τους αριθμούς: 2, 5, 7, 8, 10, 12, ...(όλοι οι ζυγοί)..., 100.

Απάντηση

Είναι λάθος να κάνουμε άσκοπες συγκρίσεις, ειδικά μέσα σε δομή επανάληψης. Μια χαζή λύση θα ήταν

```
Αλγόριθμος Αντιπαράδειγμα
Για κ από 1 μέχρι 100
  Αν κ = 2 ή κ = 5 ή κ = 7 ή (κ ≥ 8 και κ mod 2 = 0) τότε
    Εμφάνισε κ
  Τέλος_αν
Τέλος_επανάληψης
Τέλος
```

Θα πρέπει εκατό φορές να κάνουμε 1 πράξη και 5 συγκρίσεις. Η σωστή λύση:

```
Αλγόριθμος Σωστό
Εμφάνισε 2
Εμφάνισε 5
Εμφάνισε 7
Για κ από 8 μέχρι 100 με_βήμα 2
  Εμφάνισε κ
Τέλος_επανάληψης
Τέλος
```

Άσκηση 63 - κλιμακωτή χρέωση σε απλή μορφή

Μια εταιρεία κινητής τηλεφωνίας χρεώνει τουλάχιστον 30 δευτερόλεπτα ομιλίας, ακόμη κι αν ο συνδρομητής μίλησε λιγότερο. Τα πρώτα 30 δευτερόλεπτα χρεώνονται προς 0,02 € το δευτερόλεπτο και τα επόμενα προς 0,03€ το δευτερόλεπτο. Να γραφεί συνάρτηση σε ΓΛΩΣΣΑ η οποία να δέχεται ως όρισμα τον αριθμό των δευτερολέπτων που μίλησε ένας συνδρομητής και να υπολογίζει την χρέωσή του.

Απάντηση

Στην άσκηση έχουμε ελάχιστη χρέωση και κλιμακωτή χρέωση αλλά σε πολύ απλή μορφή. Επειδή η ελάχιστη χρέωση συμπίπτει με την πρώτη κλίμακα θα χρειαστούμε μόνο μια απλή επιλογή για να υπολογίσουμε την χρέωση.

```
ΣΥΝΑΡΤΗΣΗ χρέωσηΚλήσης(χρόνος): ΠΡΑΓΜΑΤΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χρόνος
  ΠΡΑΓΜΑΤΙΚΕΣ: χρέωση
ΑΡΧΗ
  ! για οποιονδήποτε χρόνο, η χρέωση είναι πάντα 30*0.02=0.6
  χρέωση <- 0.6 ! 30*0.02
  ΑΝ χρόνος > 30 ΤΟΤΕ ! αν έχουμε επιπλέον χρέωση, την προσθέτουμε
    χρέωση <- χρέωση + (χρόνος - 30)*0.05
  ΤΕΛΟΣ_ΑΝ
  χρέωσηΚλήσης <- χρέωση
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Ένα συνηθισμένο λάθος στην ελάχιστη χρέωση είναι να την περιλαμβάνουμε μέσα σε όλες τις δυνατότητες της επιλογής. Σ' αυτήν περίπτωση η δομή επιλογής θα γίνει κάπως έτσι

```
  ΑΝ χρόνος <= 30 ΤΟΤΕ
    χρέωση <- 0.6  ! 30*0.02
  ΑΛΛΙΩΣ
    χρέωση <- 0.6 + (χρόνος - 30)*0.05
  ΤΕΛΟΣ_ΑΝ
```

Ο αλγόριθμος δεν είναι λάθος, αλλά κουτός, ιδιαίτερα όταν έχουμε πολλές επιλογές. Αν κάτι συμβαίνει οπωσδήποτε, τότε η θέση του είναι έξω από το μπλοκ της **Αν**. (δείτε στην άσκηση 65 ένα πιο χαρακτηριστικό παράδειγμα)

Μια διαδεδομένη πρακτική είναι να αντικαθιστούμε μέσα στον κώδικα τις πράξεις με σταθερούς αριθμούς με το ακριβές αποτέλεσμα. Στο παρελθόν οι διερμηνευτές και μεταφραστές δεν κάνανε καμία βελτίωση στον κώδικα. Αυτό σήμαινε ότι αν μέσα σε μια επανάληψη υπήρχε ο πολλαπλασιασμός `0.1*0.1*100`, ένα εκατομμύριο φορές, ο επεξεργαστής θα υποχρεωνόταν να υπολογίσει ένα εκατομμύριο φορές το ίδιο γινόμενο. Έτσι, ήταν απαράβατος κανόνας να αντικαθιστώνται οι σταθερές πράξεις με το αποτέλεσμά τους. Οι σύγχρονες γλώσσες εντοπίζουν τέτοια σημεία του κώδικα και αντικαθιστούν από μόνες τους τις πράξεις. Η συνήθεια όμως παρέμεινε. Σε κάθε περίπτωση, όταν χρησιμοποιούμε έτοιμο αποτέλεσμα, περιλαμβάνουμε, σε μορφή σχολίου, τις πράξεις που το παράγουν.

Άσκηση 64 - κλιμακωτή χρέωση

Ίδιο με το προηγούμενο, αλλά τώρα η εταιρεία εφαρμόζει πιο πολύπλοκη πολιτική. Ο ελάχιστος χρόνος χρέωσης είναι 20 δευτερόλεπτα. Τα πρώτα 60 δευτερόλεπτα χρεώνονται προς 0.02€ ανά δευτερόλεπτο. Από το 61ο δευτερόλεπτο και μέχρι τα 3 λεπτά (180 δευτερόλεπτα) χρεώνονται προς 0.05€ και τέλος για περισσότερο από 3 λεπτά (181ο δευτερόλεπτο) η χρέωση ανεβαίνει στα 0.07€ ανά δευτερόλεπτο. Θέλουμε μια συνάρτηση γραμμένη σε ΓΛΩΣΣΑ η οποία να δέχεται ως όρισμα τον αριθμό των δευτερολέπτων που μίλησε ένας συνδρομητής και να υπολογίζει την χρέωσή του.

Απάντηση

1ος τρόπος

Ένας τρόπος να αντιμετωπίσουμε τέτοιου είδους προβλήματα είναι με μια μεγάλη δομή επιλογής με πολλά **ΑΛΛΙΩΣ_ΑΝ** (ουσιαστικά επέκταση του αλγορίθμου της προηγούμενης άσκησης). Οι συνθήκες μπαίνουν με τέτοια σειρά ώστε με έναν έλεγχο να αποφασίζεται η κλίμακα.

```
ΣΥΝΑΡΤΗΣΗ χρέωσηΚλήσης(χρόνος): ΠΡΑΓΜΑΤΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χρόνος
  ΠΡΑΓΜΑΤΙΚΕΣ: χρέωση
ΑΡΧΗ
  ! ελάχιστη χρέωση
  ΑΝ χρόνος <= 20 ΤΟΤΕ
    χρέωση <- 0.4 ! 20*0.02

  ! 1η κλιμακα
  ΑΛΛΙΩΣ_ΑΝ χρόνος <= 60 ΤΟΤΕ
    χρέωση <- χρόνος*0.02


  ! 2η κλίμακα
  ΑΛΛΙΩΣ_ΑΝ χρόνος <= 180 ΤΟΤΕ
   !χρέωση <- 60*0.02 + (χρόνος - 60)*0.05
    χρέωση <- χρόνος*0.05 - 1.8 ! το αποτέλεσμα των πιο πάνω πράξεων

  ! 3η κλίμακα
  ΑΛΛΙΩΣ
   !χρέωση <- 60*0.02 + (180 - 60)*0.05 + (χρόνος - 180)*0.07
    χρέωση <- 0.07*χρόνος - 5.4 ! το αποτέλεσμα των πιο πάνω των πράξεων
  ΤΕΛΟΣ_ΑΝ
  χρέωσηΚλήσης <- χρέωση
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

*** 2ος τρόπος**

Η προηγούμενη λύση είναι καλή για ένα εισαγωγικό μάθημα προγραμματισμού, αλλά αν δουλεύατε σε εταιρεία λογισμικού δεν θα γινόταν δεκτή. Το σοβαρό πρόβλημα του προηγούμενου αλγόριθμου είναι η επανάληψη της χρέωσης όλων των προηγούμενων κλιμάκων σε κάθε `ΑΛΛΙΩΣ_ΑΝ`. Στην πραγματικότητα επαναλαμβάνουμε κώδικα. Η επανάληψη του κώδικα εκτός από άκομψη είναι και μη πρακτική. Φανταστείτε αν αλλάζει συχνά η εταιρεία τις χρεώσεις και τις κλίμακες (όπως γίνεται συνήθως). Μια μικρή αλλαγή στην χρέωση της πρώτης κλίμακας επιφέρει αλλαγή στον κώδικα όλων των επόμενων χρεώσεων.

Ένας πιο κομψός και καθαρός τρόπος είναι να υλοποιήσουμε την κλιμακωτή χρέωση ακριβώς όπως την υπολογίζει ένας άνθρωπος. Χρεώνουμε κάθε φορά τον χρόνο της κλίμακας και αφήνουμε τον υπόλοιπο χρόνο να χρεωθεί στην επόμενη κλίμακα. Ακόμη, ο ελάχιστος χρόνος χρέωσης είναι άσχετος με την κλιμακωτή χρέωση και κανονικά πρέπει να υλοποιηθεί με χωριστό, δικό του, κώδικα.

```
ΣΥΝΑΡΤΗΣΗ χρέωσηΚλήσης2(χρόνος): ΠΡΑΓΜΑΤΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χρόνος
  ΠΡΑΓΜΑΤΙΚΕΣ: χρέωση
ΑΡΧΗ
 ! ελάχιστη χρέωση
  ΑΝ χρόνος < 20 ΤΟΤΕ
    χρόνος <- 20
  ΤΕΛΟΣ_ΑΝ

 ! 1η κλίμακα
  ΑΝ χρόνος > 60 ΤΟΤΕ
    χρέωση <- 1.2 ! 60*0.02 ! χρεώνουμε
    χρόνος <- χρόνος – 60   ! αφαιρούμε όσο χρόνο χρεώσαμε
  ΑΛΛΙΩΣ
    χρέωση <- χρόνος*0.02   ! χρεώνουμε όλο τον χρόνο
    χρόνος <- 0             ! δεν έμεινε χρόνος για την επόμενη κλίμακα
  ΤΕΛΟΣ_ΑΝ

 ! 2η κλίμακα
  ΑΝ χρόνος > 120 ΤΟΤΕ      ! το εύρος της κλίμακας, όχι η τιμή της
    χρέωση <- χρέωση + 6 ! 120*0.05
    χρόνος <- χρόνος - 120
  ΑΛΛΙΩΣ
    χρέωση <- χρέωση + χρόνος*0.05
    χρόνος <- 0
  ΤΕΛΟΣ_ΑΝ

 ! 3η κλίμακα (τελευταία)
  ΑΝ χρόνος > 0 ΤΟΤΕ        ! αν έχει μείνει χρόνος για χρέωση
    χρέωση <- χρέωση + χρόνος*0.07
    χρόνος <- 0
  ΤΕΛΟΣ_ΑΝ

  χρέωσηΚλήσης <- χρέωση
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Ο κώδικας που προκύπτει με τον δεύτερο τρόπο δεν είναι μικρότερος από τον πρώτο. Ίσως δεν φαίνεται ούτε κομψότερος με μια πρώτη ματιά. Είναι όμως πιο καθαρός επειδή δεν μπλέκει τις κλίμακες μεταξύ τους και δεν επαναλαμβάνει την λογική των χρεώσεων. Η κάθε κλίμακα δρα ανεξάρτητα από τις άλλες. Έτσι, είναι

πολύ εύκολα επεκτάσιμος. Το μόνο αρνητικό που έχει είναι ότι επαναλαμβάνει τον κώδικα της κάθε κλίμακας ξανά και ξανά. Αυτό διορθώνεται εύκολα βάζοντας τον κώδικα κάθε κλίμακας μέσα σε επανάληψη!

```
ΣΥΝΑΡΤΗΣΗ χρέωσηΚλήσης3(χρόνος): ΠΡΑΓΜΑΤΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: κ
  ΠΡΑΓΜΑΤΙΚΕΣ: χρόνος, χρέωση, Α[3, 2]
ΑΡΧΗ
  ! Ο πίνακας με τις κλίμακες. Κάθε γραμμή και μια κλίμακα.
  ! Η πρώτη στήλη είναι το εύρος, η δεύτερη η χρέωση
  Α[1, 1] <- 60        ! η 1η κλίμακα
  Α[1, 2] <- 0.02

  Α[2, 1] <- 120       ! η 2η κλίμακα
  Α[2, 2] <- 0.05

  Α[3, 1] <- 100000000 ! η τελευταία κλίμακα φτάνει στο άπειρο
  Α[3, 2] <- 0.07

  ! ελάχιστη χρέωση
  ΑΝ χρόνος < 20 ΤΟΤΕ
    χρόνος <- 20
  ΤΕΛΟΣ_ΑΝ

  χρέωση <- 0
  κ <- 1
  ΟΣΟ χρόνος > 0 ΕΠΑΝΑΛΑΒΕ
    ΑΝ χρόνος > Α[κ, 1] ΤΟΤΕ             ! αν ο χρόνος ξεπερνά το εύρος
      χρέωση <- χρέωση + Α[κ, 1]*Α[κ, 2] ! χρέωσε όλο το εύρος
      χρόνος <- χρόνος - Α[κ, 1]         ! αφαίρεσε όσο χρόνο χρεώθηκε
    ΑΛΛΙΩΣ
      χρέωση <- χρέωση + χρόνος*Α[κ, 2]  ! χρέωσε όλο τον χρόνο
      χρόνος <- 0                        ! δεν μένει τίποτα
    ΤΕΛΟΣ_ΑΝ
    κ <- κ + 1
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  χρέωσηΚλήσης <- χρέωση
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Ο κώδικάς μας πλέον έγινε επαγγελματικός. Δουλεύει για οποιοδήποτε πλήθος κλιμάκων χωρίς καμία αλλαγή. Το μόνο που χρειάζεται είναι να ορίσουμε τις χρεώσεις και το εύρος της κάθε κλίμακας στον πίνακα `Α[]`. Το μόνο παραπάνω που θα έκανε ένας επαγγελματίας είναι να διαβάζει τον πίνακα `Α[]` από ένα εξωτερικό αρχείο, ώστε να μην χρειάζεται η ελάχιστη επέμβαση στον κώδικα.

Άσκηση 65 - χρέωση υπό συνθήκη (μη κλιμακωτή)

Μια εταιρεία χορηγεί επίδομα σπουδών στους υπαλλήλους της με βάση τις γραμματικές τους γνώσεις σαν ποσοστό του βασικού τους μισθού ως εξής:

(i) για τους αποφοίτους γυμνασίου 4%
(ii) για τους αποφοίτους λυκείου 8%
(iii) για τους πτυχιούχους ανώτατης εκπαίδευσης 20%

Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να διαβάζει τον βασικό μισθό και τον κωδικό σπουδών (1, 2 ή 3 για τα τρία επίπεδα σπουδών) ενός υπαλλήλου και να υπολογίζει το επίδομα σπουδών που θα του χορηγηθεί.

Απάντηση

Όταν η χρέωση δεν είναι κλιμακωτή τα πράγματα είναι πολύ πιο εύκολα.

```
Αλγόριθμος ΕπίδομαΣπουδών1
Εμφάνισε "Δώστε το επίπεδο γνώσεων (1 γυμν, 2 λύκ, 3 ανώτ)"
Διάβασε επίπεδο
Εμφάνισε "Δώστε τον μισθό"
Διάβασε μισθός
Αν επίπεδο = 1 τότε
  επίδομα ← μισθός* 0.04
αλλιώς_αν επίπεδο = 2 τότε
  επίδομα ← μισθός* 0.06
αλλιώς_αν επίπεδο = 3 τότε
  επίδομα ← μισθός* 0.2
Τέλος_αν
Εμφάνισε "Το επίδομα σπουδών είναι :", επίδομα
Τέλος
```

Ο κώδικας φαίνεται τέλειος, εκτός από το γεγονός ότι ο **μισθός** επαναλαμβάνεται σε όλες τις επιλογές της δομής **Αν**. Είναι λάθος να περιλαμβάνουμε σε μια δομή επιλογής την ίδια λογική σε όλες τις συνθήκες. Θα χρησιμοποιήσουμε μία ακόμη μεταβλητή και θα γράψουμε τον κώδικα έτσι:

```
...
Διάβασε μισθός
Αν επίπεδο = 1 τότε
  ποσοστό ← 0.04
αλλιώς_αν επίπεδο = 2 τότε
  ποσοστό ← 0.06
αλλιώς_αν επίπεδο = 3 τότε
  ποσοστό ← 0.2
Τέλος_αν
επίδομα ← ποσοστό * μισθός ...
```

*Μην περιλαμβάνετε μέσα σε **Αν** τον ίδιο κώδικα ξανά και ξανά. Αν κάτι πρέπει να εκτελεστεί έτσι κι αλλιώς τότε πρέπει να βρίσκεται έξω από το μπλοκ της **Αν**. Ο σκοπός της **Αν** είναι να διαλέξει αυτό που είναι διαφορετικό όχι αυτό που είναι ίδιο.*

Το κόστος μιας ακόμη μεταβλητής είναι πολύ μικρό μπροστά στο κέρδος του καθαρού κώδικα. Ο πιο καθαρός κώδικας πολλές φορές γίνεται και πιο μικρός. Αφού απλοποιήσουμε την **Αν** γίνεται ξεκάθαρος ο σκοπός της. Το μόνο που κάνει είναι να διαλέγει το σωστό ποσοστό. Πετυχαίνουμε την ίδια λειτουργικότητα θεωρώντας τη μεταβλητή **επίπεδο** ως δείκτη σε έναν πίνακα κι έτσι η **Αν** είναι εντελώς περιττή.

```
Αλγόριθμος ΕπίδομαΣπουδών2
Εμφάνισε "Δώστε το επίπεδο γνώσεων (1 γυμν, 2 λύκ, 3 ανώτ): "
Διάβασε επίπεδο
Εμφάνισε "Δώστε τον μισθό: "
Διάβασε μισθός

A[1] ← 0.04
A[2] ← 0.06
A[3] ← 0.2
επίδομα ← μισθός* A[επίπεδο]

Εμφάνισε "Το επίδομα σπουδών είναι :", επίδομα
Τέλος
```

Άσκηση 66 - σπάστε το πρόγραμμα σε διαδικασίες

Μια Τράπεζα ακολουθεί την εξής διαδικασία κατά τη διαδικασία ανάληψης χρημάτων μέσω ενός μηχανήματος ΑΤΜ: ο πελάτης καταχωρεί τον μυστικό αριθμό πρόσβασης (PIN) και αν γίνει λάθος καταχώρηση έως και 3 φορές, το μηχάνημα κρατάει την κάρτα του πελάτη. Ακόμη, το μέγιστο ποσό που μπορεί να κάνει ανάληψη ένας πελάτης σε μια συναλλαγή του είναι 500 €.

(α) Να διαβασθούν ο μυστικός αριθμός PIN ενός πελάτη και το υπόλοιπο που υπάρχει στον λογαριασμό του (υπολ)

(β) Να ελεγχθεί αν γίνεται σωστά η καταχώρηση του PIN έτσι ώστε το μηχάνημα να του επιτρέψει να συνεχίσει τη συναλλαγή ή θα του κρατήσει την κάρτα

(γ) Σε περίπτωση που του επιτρέψει να συνεχίσει τη συναλλαγή, να διαβάζει το ποσό που επιθυμεί να κάνει ανάληψη ο πελάτης, να ελέγχει αν είναι μεγαλύτερο από το μέγιστο όριο ή από το υπόλοιπο που υπάρχει στον λογαριασμό του και όταν το ποσό προς ανάληψη είναι αποδεκτό, να υπολογίζει και να εμφανίζει το διαθέσιμο υπόλοιπο του λογαριασμού του πελάτη.

Απάντηση

Σπάστε το πρόγραμμα σε διαδικασίες είτε το ζητάει η εκφώνηση είτε όχι. Προσπαθήστε να σκεφτείτε τι χρειάζεται να ξέρει η διαδικασία την στιγμή που την καλείτε και τι πρέπει να επιστρέψει στο κυρίως πρόγραμμα. Εμφανίστε τις διαδικασίες με την σειρά που τις ζητά η εκφώνηση.

(α)

```
ΔΙΑΔΙΚΑΣΙΑ Διάβασε_ΠΙΝ_Υπολοιπο(ΠΙΝ, υπολ)
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: ΠΙΝ
  ΠΡΑΓΜΑΤΙΚΕΣ: υπολ
ΑΡΧΗ
  ! υποτίθεται στο σημείο αυτό ότι διαβάζουμε από μία
  ! βάση δεδομένων, δίνοντας τον κωδικό πελάτη
  ΔΙΑΒΑΣΕ ΠΙΝ
  ΔΙΑΒΑΣΕ υπολ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

(β)

```
ΔΙΑΔΙΚΑΣΙΑ ΈλεγχοςΠΙΝ(ΠΙΝ, κωδικόςΣωστός)
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: προσπ, ΠΙΝ, εισαγΠΙΝ
  ΛΟΓΙΚΕΣ: κωδικόςΣωστός
ΑΡΧΗ
  προσπ <- 0              ! μετράει τις προσπάθειες
  κωδικόςΣωστός <- ΨΕΥΔΗΣ ! σημαία που παρακολουθεί τον κωδικό
  ΟΣΟ προσπ < 3 ΚΑΙ ΟΧΙ κωδικόςΣωστός ΕΠΑΝΑΛΑΒΕ
    προσπ <- προσπ + 1
    ΓΡΑΨΕ "Εισάγετε PIN (προσπάθεια ", προσπ, " από 3)"
    ΔΙΑΒΑΣΕ εισαγΠΙΝ
    ΑΝ ΠΙΝ = εισαγΠΙΝ ΤΟΤΕ
      κωδικόςΣωστός <- ΑΛΗΘΗΣ ! πιο καλό: κωδικόςΣωστός <- ΠΙΝ = εισαγΠΙΝ
    ΤΕΛΟΣ_ΑΝ
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

! αν έχουμε βγει από την επανάληψη και ο χρήστης δεν εισήγαγε..
! ..σωστό κωδικό, τότε έχει ξεπεράσει τις τρεις προσπάθειες
  ΑΝ ΟΧΙ κωδικόςΣωστός ΤΟΤΕ
    ΓΡΑΨΕ "Η κάρτα σας κλειδώθηκε"
```

```
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

(γ)

```
ΔΙΑΔΙΚΑΣΙΑ ΚάνεΑνάληψη(υπολ)
ΜΕΤΑΒΛΗΤΕΣ
  ΠΡΑΓΜΑΤΙΚΕΣ: υπολ, ανάληψη
ΑΡΧΗ
  ΓΡΑΨΕ "Δώστε το ποσό της ανάληψης"
  ΔΙΑΒΑΣΕ ανάληψη
  ΑΝ ανάληψη <= υπολ ΤΟΤΕ
    ΓΡΑΨΕ "Αποδεκτή ανάληψη. Νέο υπόλοιπο: ", υπολ - ανάληψη
  ΑΛΛΙΩΣ
    ΓΡΑΨΕ "Μη επαρκές υπόλοιπο. Η διαδικασία ακυρώθηκε."
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Στο τέλος μπορούμε εύκολα να συνδέσουμε τις διαδικασίες σε ένα κυρίως πρόγραμμα

```
ΠΡΟΓΡΑΜΜΑ ΑΤΜ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: ΠΙΝ
  ΠΡΑΓΜΑΤΙΚΕΣ: υπολ
  ΛΟΓΙΚΕΣ: ΠΙΝσωστό
ΑΡΧΗ
  ΚΑΛΕΣΕ Διάβασε_ΠΙΝ_Υπολοιπο(ΠΙΝ, υπολ)
  ΚΑΛΕΣΕ ΈλεγχοςΠΙΝ(ΠΙΝ, ΠΙΝσωστό)
  ΑΝ ΠΙΝσωστό ΤΟΤΕ
    ΚΑΛΕΣΕ ΚάνεΑνάληψη(υπολ)
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Άσκηση 67 - έγκυρο ΑΦΜ

Ένας Αριθμός Φορολογικού Μητρώου (ΑΦΜ) θεωρείται έγκυρος άν πληρεί ορισμένες προϋποθέσεις. Έστω ο ΑΦΜ $A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9$, όπου A_i το i-στό ψηφίο. Ο αλγόριθμος ελέγχου εγκυρότητας είναι ο ακόλουθος

1) Βρίσκουμε το άθροισμα
$\Sigma = A_1 \cdot 256 + A_2 \cdot 128 + A_3 \cdot 64 + A_4 \cdot 32 + A_5 * 16 + A_6 \cdot 8 + A_7 \cdot 4 + A_8 * 2$. Το τελευταίο ψηφίο δεν συμμετέχει στο άθροισμα (είναι το ψηφίο ελέγχου).

2) Υπολογίζουμε το υπόλοιπο της διαίρεσης του Σ με τον αριθμό 11.

3) Αν το υπόλοιπο της διαίρεσης είναι 10, τότε το A_9 πρέπει να είναι ίσο με μηδέν. Σε αντίθετη περίπτωση, πρέπει το υπόλοιπο να είναι ίσο με το A_9.

Να γραφεί συνάρτηση σε ΓΛΩΣΣΑ, **έγκυροΑΦΜ(χ)**, όπου **χ** ένας εννια-ψήφιος ακέραιος θετικός αριθμός (ο οποίος θα αναπαριστά το ΑΦΜ) η οποία να ελέγχει αν είναι έγκυρο ή όχι. Η συνάρτηση πρέπει να επιστρέφει **Αληθής** αν ο αριθμός πληρεί όλες τις προϋποθέσεις για να είναι έγκυρο ΑΦΜ και **Ψευδής** σε κάθε άλλη περίπτωση (π.χ. αρνητικού αριθμού, μη 9-ψήφιου αριθμού κτλ)

Απάντηση

Τα ψηφία του αριθμού (εκτός του τελευταίου) πολλαπλασιάζονται διαδοχικά με τις δυνάμεις του 2, με εκθέτες από 1 έως 8. Έτσι ο υπολογισμός του αθροίσματος μπορεί να γίνει με επανάληψη. Τα βήματα 2 και 3 του αλγόριθμου μπορούν να γίνουν ένα βήμα. Να υπολογίσουμε το υπόλοιπο της διαίρεσης με το 11 και ότι βρούμε ξανά το υπόλοιπο με το 10. Αυτό πρέπει να είναι το τελευταίο ψηφίο του ΑΦΜ.

Επειδή η ΓΛΩΣΣΑ θεωρεί την ύψωση σε δύναμη πάντα πραγματικό αριθμό, πρέπει να γράψουμε μια δική μας συνάρτηση που να κάνει ύψωση σε δύναμη ακεραίου σε ακέραιο και να επιστρέφει ακέραιο.

```
ΣΥΝΑΡΤΗΣΗ ΈγκυροΑΦΜ(αφμ): ΛΟΓΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: αφμ, κ, Σ, ψηφίο, τελευταίο
ΑΡΧΗ
  τελευταίο <- αφμ mod 10
  αφμ <- αφμ div 10
  Σ <- 0
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 8
    ψηφίο <- αφμ mod 10
    Σ <- Σ + ψηφίο* ΑκέραιαΔύναμη(2, κ)
    αφμ <- αφμ div 10
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΈγκυροΑΦΜ <- (Σ mod 11) mod 10 = τελευταίο
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ

ΣΥΝΑΡΤΗΣΗ ΑκέραιαΔύναμη(βάση, εκθέτης): ΑΚΕΡΑΙΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: βάση, εκθέτης, κ, γινόμενο
ΑΡΧΗ
  γινόμενο <- 1
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ εκθέτης
    γινόμενο <- γινόμενο* βάση
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΑκέραιαΔύναμη <- γινόμενο
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Άσκηση 68 - παλίνδρομος αριθμός

Παλίνδρομος αριθμός είναι ένας φυσικός αριθμός που διαβάζεται το ίδιο από αριστερά προς τα δεξιά και από τα δεξιά προς τα αριστερά (ανάποδα). Για παράδειγμα οι αριθμοί 125434521 και 3773 είναι παλίνδρομοι.

(α) Να γραφεί συνάρτηση `Παλιν1(x)` σε ΓΛΩΣΣΑ που να δέχεται ως όρισμα έναν φυσικό τετραψήφιο αριθμό και να επιστρέφει την λογική τιμή `Αληθής` αν αυτός είναι παλίνδρομος ή `Ψευδής` σε αντίθετη περίπτωση. Δεν χρειάζεται να κάνετε έλεγχο δεδομένων. Θεωρείστε ότι η συνάρτηση καλείται πάντα με τετραψήφιο φυσικό αριθμό.

* (β) Γράψτε μια δεύτερη συνάρτηση `Παλιν2(x)` η οποία να δουλεύει όπως η προηγούμενη, αλλά τώρα για οποιονδήποτε φυσικό αριθμό και όχι μόνο για τετραψήφιο (για οσαδήποτε ψηφία)

Απάντηση

(α)

```
ΣΥΝΑΡΤΗΣΗ Παλίν1(χ): ΛΟΓΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χ
  ΛΟΓΙΚΕΣ: παλίνδρομος
ΑΡΧΗ
  παλίνδρομος <- χ mod 10 = χ div 1000 ! τσεκάρουμε πρώτο με τελευταίο
  ΑΝ παλίνδρομος ΤΟΤΕ                  ! αν πήγαν όλα καλά
    χ <- χ mod 1000                    ! κόψε το πρώτο ψηφίο
    χ <- χ div 10                      ! κόψε το τελευταίο ψηφίο
    παλίνδρομος <- χ mod 10 = χ div 10 ! τσέκαρε τα δύο μεσαία
```

```
  ΤΕΛΟΣ_ΑΝ
  Παλίν1 <- παλίνδρομος
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

(β) Θα χρειαστούμε την συνάρτηση `ΠλήθοςΨηφίων(χ)` από την άσκηση 27 και την `ΑκέραιαΔύναμη(βάση, εκθέτης)` από την άσκηση 67.

```
ΣΥΝΑΡΤΗΣΗ Παλίν2(χ): ΛΟΓΙΚΗ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: χ, ψηφία, πρώτοΨηφίο, τελευταίοΨηφίο
  ΛΟΓΙΚΕΣ: παλίνδρομος
ΑΡΧΗ
  ψηφία <- ΠλήθοςΨηφίων(χ)
  παλίνδρομος <- ΑΛΗΘΗΣ
  ΟΣΟ παλίνδρομος ΚΑΙ χ > 0 ΕΠΑΝΑΛΑΒΕ
    ! ο έλεγχος
    πρώτοΨηφίο <- χ div ΑκέραιαΔύναμη(10, ψηφία-1)
    τελευταίοΨηφίο <- χ mod 10
    παλίνδρομος <- πρώτοΨηφίο = τελευταίοΨηφίο

    ! προετοιμασία για την επόμενη επανάληψη
    χ <- χ mod ΑκέραιαΔύναμη(10, ψηφία-1) ! κόψε το πρώτο ψηφίο
    χ <- χ div 10                          ! κόψε το τελευταίο ψηφίο
    ψηφία <- ψηφία - 2                     ! τα ψηφία μειώθηκαν κατά 2
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  Παλίν2 <- παλίνδρομος
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Άσκηση 69 - 24ωρη μορφή σε 12ωρη

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να διαβάζει την ώρα σε 24ωρη μορφή, σαν έναν τετραψήφιο αριθμό, όπως για παράδειγμα 1452. Πρώτα να γίνει έλεγχος αν παριστάνει σωστή ένδειξη ώρας. Αν δεν παριστάνει έγκυρη ώρα να εμφανίζεται κατάλληλο μήνυμα. Αν είναι έγκυρη ώρα τότε να εμφανίζεται στην οθόνη σε 12ωρη μορφή, για παράδειγμα 2:52μμ.

Απάντηση

Αρχικά σπάμε τον τετραψήφιο αριθμό στα δύο πρώτα και δύο τελευταία ψηφία. Ο έλεγχος για έγκυρη ώρα είναι προφανής. Ο αλγόριθμος για την μετατροπή σε 12ωρη μορφή είναι να αντικαταστήσουμε την ώρα με το υπόλοιπο της διαίρεσης με το 12. Τα λεπτά παραμένουν ίδια. Μόνη εξαίρεση αποτελεί η ώρα 12:00, η οποία παραμένει 12:00 και δεν μετατρέπεται σε 00:00.

```
Αλγόριθμος ΩραΣε12ωρη
Εμφάνισε "Δώστε την ώρα"
Διάβασε χ
ώρα ← χ div 100   ! κόβουμε τα δύο τελευταία ψηφία
λεπτά ← χ mod 100 ! κόβουμε τα δύο πρώτα ψηφία

Αν χ < 0 ή ώρα > 23 ή λεπτά > 59 τότε !καλύπτουμε την περίπτωση αρνητικού
  Εμφάνισε "μη έγκυρη ώρα"
αλλιώς_αν ώρα = 12 τότε        ! 12:00 το μεσημέρι είναι ειδική περίπτωση
  Εμφάνισε "12:", λεπτά, " "
αλλιώς
  Εμφάνισε ώρα mod 12, ":", λεπτά, " "
Τέλος_αν

! τέλος προσθέτουμε το πμ ή μμ
Αν ώρα < 12 τότε
```

```
  Εμφάνισε “πμ”
Αλλιώς
  Εμφάνισε “μμ”
Τέλος_Αν
Τέλος
```

Άσκηση 70 - αγώγι με αεροπλάνο

Ένα αεροπλάνο έχει τη δυνατότητα να μεταφέρει σε κάθε πτήση μέχρι 100 τόνους εμπορεύματα σε μορφή κιβωτίων. Θεωρείστε ότι όλα τα κιβώτια έχουν το ίδιο μέγεθος και το αεροπλάνο χωράει άπειρα κιβώτια αρκεί αυτά να μην ξεπερνούν σε βάρος τους 100 τόνους.

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ρωτά τα βάρη 20 πακέτων σε τόνους (το κάθε πακέτο έχει λιγότερο βάρος από 100 τόνους). Μόλις συμπληρωθεί μία πτήση να εμφανίζει μήνυμα με το βάρος της πτήσης και στο τέλος να εμφανίζει πόσες ήταν συνολικά οι πτήσεις.

Απάντηση

```
Αλγόριθμος ΓέμισμαΠτήσεων
πτήσεις ← 0
συνΒάρος ← 0 ! το συνολικό βάρος
Για κ από 1 μέχρι 20
  Εμφάνισε "Δώστε το βάρος του επόμενου κιβωτίου"
  Διάβασε βάρος
  Αν συνΒάρος + βάρος ≤ 100 τότε ! αν το κιβώτιο χωράει
    συνΒάρος ← συνΒάρος + βάρος  ! το προσθέτουμε
  αλλιώς                         ! αν δεν χωράει
    πτήσεις ← πτήσεις + 1        ! τότε έχουμε νέα πτήση
    Εμφάνισε "νέα πτήση, μέ βάρος: ", συνΒάρος
    συνΒάρος ← βάρος             ! το νέο συνολικό βάρος είναι..
  Τέλος_αν                       ! ..το κιβώτιο που δεν χώρεσε
Τέλος_επανάληψης

Αν συνΒάρος > 0 τότε
  πτήσεις ← πτήσεις + 1      ! η τελευταία πτήση παίρνει ότι περίσσεψε
Τέλος_Αν

Εμφάνισε "Αριθμός πτήσεων: ", πτήσεις
Τέλος
```

Άσκηση 71 - άθροισμα τετραγώνων 1 έως n

Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να ζητάει έναν ακέραιο αριθμό n και να υπολογίζει το άθροισμα των τετραγώνων όλων των φυσικών αριθμών από 1 έως n, δηλαδή το άθροισμα $1^2+2^2+...+n^2$. Το αποτέλεσμα να εμφανίζεται στην οθόνη.

Γράψτε έναν δεύτερο αλγόριθμο που να υπολογίζει το ίδιο άθροισμα με τον έτοιμο τύπο από τα μαθηματικά: $\sum_{i=1}^{n} i^2=\frac{1}{6}n(n+1)(2n+1)$. Ποιον από τους δύο αλγορίθμους θα προτιμούσατε;

Απάντηση

Αν υποθέσουμε ότι δεν ξέρουμε τον μαθηματικό τύπο

```
Αλγόριθμος ΆθροισμαΤετραγώνων1
Διάβασε n
αθρ ← 0
Για i από 1 μέχρι n
```

```
  αθρ ← αθρ + i^2
Τέλος_επανάληψης
Εμφάνισε "Το άθροισμα των τετραγώνων είναι: ", αθρ
Τέλος
```

Αν χρησιμοποιήσουμε τον τύπο

```
Αλγόριθμος ΆθροισμαΤετραγώνων2
Διάβασε n
Εμφάνισε "Το άθροισμα των τετραγώνων είναι: ", n*(n + 1)*(2*n + 1)/6.0
Τέλος
```

Ασφαλώς ο δεύτερος αλγόριθμος είναι πολύ πιο αποδοτικός αφού βρίσκει το αποτέλεσμα με μερικές πράξεις χωρίς καθόλου επανάληψη. Η διαφορά είναι πιο έντονη όσο πιο μεγάλο είναι το n.

Άσκηση 72 - πρόσθεση μεγάλων ακεραίων

Σε όλες τις γλώσσες προγραμματισμού μπορούμε να προσθέσουμε δύο ακεραίους χωρίς την χρήση ειδικού αλγορίθμου. Υπάρχει όμως περιορισμός στο μέγεθος των ακεραίων που μπορεί να χειριστεί η γλώσσα προγραμματισμού και η αρχιτεκτονική του υπολογιστή μας. Για τους σύγχρονους υπολογιστές και για θετικούς μόνο ακέραιους το συνηθισμένο όριο είναι περίπου 10^{19} (~2^{64}). Ικανοποιητικά μεγάλο όριο για τις περισσότερες εφαρμογές, αλλά όχι για όλες.

```
  6253    6253    1 6253   11 6253   11 6253
+3974   +3974   +3974   +3974   +3974
              7      27     227   10227
```

Υπάρχουν εφαρμογές (πχ κρυπτογράφηση) που απαιτούν πράξεις με πραγματικά μεγάλους αριθμούς (εκατοντάδες ή και χιλιάδες ψηφία). Σε αυτές τις περιπτώσεις οι αριθμοί αναπαριστώνται με πίνακες. Το κάθε κελί του πίνακα περιέχει ένα μόνο ψηφίο του αριθμού, με φθίνουσα σειρά σημαντικότητας (όπως τους γράφουμε στο χαρτί). Οι πράξεις μεταξύ των αριθμών-πινάκων πρέπει να υλοποιηθούν με δικό μας αλγόριθμο αφού η ενσωματωμένες πράξεις της γλώσσας δεν προσθέτουν πίνακες.

Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ που να ζητάει δύο τετραψήφιους φυσικούς αριθμούς, ψηφίο-ψηφίο. Τα ψηφία που δίνει ο χρήστης να αποθηκεύονται σε δύο πίνακες `Α[4]` και `Β[4]`. Έπειτα το πρόγραμμα να προσθέτει τους δύο αριθμούς, με τον συνηθισμένο αλγόριθμο της πρόσθεσης που ξέρουμε από το Δημοτικό Σχολείο και το αποτέλεσμα να το αποθηκεύει σε έναν τρίτο πίνακα `ΑΠ[5]` (το άθροισμα δύο n-ψήφιων αριθμών δεν μπορεί να έχει περισσότερα από n+1 ψηφία).

Ο αλγόριθμος της πρόσθεσης έχει ως εξής: για να βρούμε ένα ψηφίο του αποτελέσματος, προσθέτουμε πρώτα τα δύο ψηφία των δύο προσθετέων συν το κρατούμενο. Το ψηφίο που θέλουμε είναι το υπόλοιπο της διαίρεσης του αθροίσματος με το 10, ενώ το νέο κρατούμενο είναι το πηλίκο της ίδιας διαίρεσης. Το αρχικό κρατούμενο είναι μηδέν.

Απάντηση

Επειδή θα χρειαστεί να εισάγουμε δύο αριθμούς, προτιμούμε να γράψουμε μια διαδικασία για την εισαγωγή και να την καλέσουμε δύο φορές. Αλλιώς θα πρέπει να επαναλάβουμε κώδικα (η επανάληψη κώδικα θεωρείται το προπατορικό αμάρτημα του προγραμματισμού επειδή είναι σκανδαλιστικά ελκυστικό και αργά ή γρήγορα όλοι θα υποπέσουν σ' αυτό).

```
ΠΡΟΓΡΑΜΜΑ ΠρόσθεσηΜεγάλωνΑκεραίων
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Α[4], Β[4], ΑΠ[5], κ, κρατ, ψηφίο
ΑΡΧΗ
  ΓΡΑΨΕ “Εισαγωγή 1ου αριθμού”
  ΚΑΛΕΣΕ Εισαγωγή(Α)
  ΓΡΑΨΕ “Εισαγωγή 2ου αριθμού”
  ΚΑΛΕΣΕ Εισαγωγή(Β)

  κρατ <- 0                       ! αρχικοποίηση κρατούμενου
  ΓΙΑ κ ΑΠΟ 4 ΜΕΧΡΙ 1 ΜΕ_ΒΗΜΑ -1 ! από το τέλος των αριθμών προς την αρχή
    ψηφίο <- Α[κ] + Β[κ] + κρατ
    κρατ <- ψηφίο div 10          ! το νέο κρατούμενο
    ΑΠ[κ + 1] <- ψηφίο mod 10     ! ο πίνακας ΑΠ έχει 1 θέση παραπάνω
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
  ΑΠ[1] <- κρατ                   ! το πρώτο στοιχείο του ΑΠ είναι το ..
                                  ! ..τελευταίο κρατούμενο

  ΓΡΑΨΕ ΑΠ[1], ΑΠ[2], ΑΠ[3], ΑΠ[4], ΑΠ[5]

ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ

ΔΙΑΔΙΚΑΣΙΑ Εισαγωγή(Χ) ! εισάγει έναν 4ψήφιο αριθμό στον πίνακα Χ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Χ[4], κ
ΑΡΧΗ
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 4
    ΓΡΑΨΕ "Δώστε το ", κ, " ψηφίο του αριθμού"
    ΔΙΑΒΑΣΕ Χ[κ]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Άσκηση 73 - μέτρημα λέξεων (διαδικασία προσπέρασης)

Σε έναν μονοδιάστατο πίνακα `N` θέσεων έχουμε αποθηκευμένο ένα κείμενο χαρακτήρα-χαρακτήρα, δηλαδή σε κάθε θέση του πίνακα είναι αποθηκευμένος ένας μόνο χαρακτήρας του κειμένου. Θέλουμε να μετρήσουμε τις λέξεις του κειμένου. Κάθε λέξη χωρίζεται από την επόμενη με έναν κενό διάστημα (`' '` ή `“ “`). Να γραφεί αλγόριθμος σε ψευδογλώσσα που να μετρά τις λέξεις για δύο περιπτώσεις:
(α) Το κείμενο είναι προσεκτικά γραμμένο ώστε μεταξύ των λέξεων να υπάρχει ένα και μόνο ένα κενό διάστημα.
(β) Το κείμενο είναι απρόσεκτα γραμμένο και έτσι μερικές φορές υπάρχουν δύο ή περισσότερα κενά διαστήματα από λέξη σε λέξη.
Και στις δύο περιπτώσεις δεν υπάρχουν σημεία στίξης (ή αν υπάρχουν είναι κολλημένα στο τέλος της λέξης, όπως είναι το σωστό). Επίσης δεν υπάρχουν παραπανίσια κενά διαστήματα στην αρχή και στο τέλος του κειμένου.

Απάντηση

(α) Αφού το κείμενο είναι γραμμένο έτσι ώστε σε κάθε λέξη να αντιστοιχεί ένα και μόνο ένα κενό, τότε αρκεί να μετρήσουμε τα κενά. Η τελευταία λέξη δεν έχει κενό (αφού δεν υπάρχουν παραπανίσια κενά), έτσι στο πλήθος των κενών που θα βρούμε θα προσθέσουμε 1 για την τελευταία λέξη

```
λέξεις ← 1              ! μετράμε την τελευταία λέξη η οποία δεν έχει κενό
Για κ από 1 μέχρι N-1   ! αποκλείεται να έχουμε κενό στο τέλος
  Αν Α[κ] = ' ' τότε    ! κάθε λέξη εκτός της τελευταίας έχει ένα κενό
    λέξεις ← λέξεις + 1
```

```
  Τέλος_αν
Τέλος_επανάληψης
```
(β) Αν υπάρχουν περισσότερα από ένα κενά μεταξύ των λέξεων τότε χρησιμοποιούμε την **διαδικασία προσπέρασης** (*skip over process*). Ο βασικός αλγόριθμος παραμένει ίδιος όπως πριν αλλά τώρα κάθε φορά που βρίσκουμε ένα κενό, προσπερνάμε όλα τα επόμενα κενά.
Η διαδικασία προσπέρασης είναι ένα απλό `Όσο-επανάλαβε` το οποίο αυξάνει τον μετρητή κατά ένα όσο διαρκεί η προσπέραση, δηλαδή
```
! διαδικασία προσπέρασης
Όσο Α[κ] = ' ' επανάλαβε
  κ ← κ + 1
Τέλος_επανάληψης
```
Προσέξτε ότι η διαδικασία προσπέρασης θα φέρει τον δείκτη κ στο πρώτο γράμμα που δεν είναι κενό. Αν εφαρμοστεί σε γράμμα που δεν είναι κενό, τότε θα αφήσει τον δείκτη στην ίδια θέση.
Εδώ έχουμε ένα θαυμάσιο παράδειγμα περίπτωσης στην οποία χρειάζεται να αλλάζουμε τον μετρητή μέσα στο σώμα της `Για-από-μέχρι`. Αν η γλώσσα μας επιτρέπει κάτι τέτοιο (όλες οι καλές γλώσσες το επιτρέπουν) τότε η ενδεδειγμένη λύση είναι
```
λέξεις ← 1
Για κ από 1 μέχρι Ν-1
  Αν Α[κ] = ' ' τότε          ! αν βρεις κενό
    λέξεις ← λέξεις + 1       ! μέτρα μία λέξη
    κ ← κ + 1                 ! προχώρα στην επόμενη θέση
    Όσο Α[κ] = ' ' επανάλαβε ! προσπέρνα όλα τα υπόλοιπα κενά
      κ ← κ + 1
    Τέλος_επανάληψης
  Τέλος_αν
Τέλος_επανάληψης
```
Δυστυχώς η πρωτόγονη PASCAL πάνω στην οποία στηρίζεται η ΓΛΩΣΣΑ, δεν επιτρέπει αλλαγή του μετρητή μέσα στην `Για-από-μέχρι` (και οι οδηγίες του υπουργείου είναι ότι δεν επιτρέπεται), συνεπώς ο τελικός αλγόριθμος είναι
```
λέξεις ← 1
κ ← 0
Όσο κ < Ν-2 επανάλαβε
  κ ← κ + 1
  Αν Α[κ] = ' ' τότε
    λέξεις ← λέξεις + 1
    κ ← κ + 1
    Όσο Α[κ] = ' ' επανάλαβε ! προσπέραση επιπλέον κενών
      κ ← κ + 1
    Τέλος_επανάληψης
  Τέλος_αν
Τέλος_επανάληψης
```

Αναζήτηση - Ταξινόμηση

Άσκηση 74 - αναζήτηση μόνο όταν χρειάζεται
Να γραφεί αλγόριθμος σε ψευδογλώσσα που να εμφανίζει στην οθόνη όλα τα πολλαπλάσια του 3 που είναι μικρότερα από 1000.

Απάντηση

Συνηθισμένο λάθος στον προγραμματισμό είναι να χρησιμοποιούμε τον αλγόριθμο αναζήτησης ακόμη κι όταν δεν ψάχνουμε τίποτα. Για παράδειγμα ο παρακάτω αλγόριθμος είναι λάθος

```
Για κ από 1 μέχρι 1000
  Αν κ mod 3 = 0 τότε Εμφάνισε κ
Τέλος_επανάληψης
```

Ο αλγόριθμος θα σαρώσει όλους τους φυσικούς μέχρι το 1000 σαν να ψάχνει κάτι. Στην πραγματικότητα δεν ψάχνουμε τίποτα. Τα πολλαπλάσια του 3 τα ξέρουμε. Είναι όλοι οι φυσικοί **κ*3**, αρκεί να μην ξεπερνούν το 1000. Ο σωστός αλγόριθμος

```
κ ← 1
Όσο κ*3 < 1000 επανάλαβε
  Εμφάνισε κ*3
  κ ← κ + 1
Τέλος_επανάληψης
```

Τώρα η **Όσο** θα επαναληφθεί τόσες φορές όσα είναι τα πολλαπλάσια του 3.

Άσκηση 75

Οι φυσικοί αριθμοί που είναι πολλαπλάσια του 5 ή του 3 και μικρότεροι του 10 είναι οι 3, 5, 6, 9. Αν τους προσθέσουμε βρίσκουμε 23. Βρείτε το άθροισμα των πολλαπλασίων του 3 ή του 5 που είναι μικρότεροι από το 1000.

Απάντηση

Θα βρούμε το άθροισμα πρώτα των πολλαπλασίων του 3 και μετά του 5. Το μόνο πρόβλημα είναι ότι κάποιοι φυσικοί μπορεί να είναι και πολλαπλάσια του 3 και του 5. Ο πιο γρήγορος τρόπος να αποφύγουμε την διπλή πρόσθεση είναι στην δεύτερη **Όσο** να αποκλείσουμε τα πολλαπλάσια του 3

```
αθρ ← 0
! τα πολλαπλάσια του 3
κ ← 1
Όσο κ*3 < 1000 επανάλαβε
  αθρ ← αθρ + κ*3
  κ ← κ + 1
Τέλος_επανάληψης

! τα πολλαπλάσια του 5
κ ← 1
Όσο κ*5 < 1000 επανάλαβε
  Αν κ*5 mod 3 ≠ 0 τότε αθρ ← αθρ + κ*5
  κ ← κ + 1
Τέλος_επανάληψης

Εμφάνισε αθρ
```

Σημείωση: μπορούμε να επιταχύνουμε λίγο τον αλγόριθμο αν αποφύγουμε τον ίδιο πολλαπλασιασμό τρεις φορές στο δεύτερο **Όσο**. Μπορούμε να υπολογίζουμε το αποτέλεσμα του πολλαπλασιασμού μία φορά για κάθε **κ**, ως εξής:

```
! τα πολλαπλάσια του 5
κ ← 1
γιν ← 5
Όσο γιν < 1000 επανάλαβε
  Αν γιν mod 3 ≠ 0 τότε αθρ ← αθρ + γιν
  κ ← κ + 1
  γιν ← κ*5
Τέλος_επανάληψης
```

Άσκηση 76 - ταξινόμηση 3 στοιχείων

Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να ταξινομεί τρεις αριθμούς. Οι αριθμοί είναι αρχικά αποθηκευμένοι στις θέσεις α, β, γ με τυχαία σειρά. Στο τέλος του αλγορίθμου πρέπει ο μικρότερος αριθμός να είναι αποθηκευμένος στο α, ο αμέσως μεγαλύτερος στο β και ο πιο μεγάλος απ' όλους στο γ.

Απάντηση

Δεν υπάρχει “έξυπνος” τρόπος για να ταξινομήσουμε τρεις αριθμούς. Θα χρειαστούμε το λιγότερο τρεις συγκρίσεις. Υπάρχουν πολλοί τρόποι. Ένας από τους καθαρότερους και πιο δημοφιλείς είναι εμπνευσμένος από το bubblesort

```
Αλγόριθμος ΤαξινόμησηΤαριθμών
Διάβασε α
Διάβασε β
Διάβασε γ

Αν α > β τότε Αντιμετάθεσε α, β ! ταξινόμησε τους 2 πρώτους

Αν β > γ τότε Αντιμετάθεσε β, γ ! ταξινόμησε του 2 τελευταίους

! στο σημείο αυτό ο μεγαλύτερος από τους 3 έχει τοποθετηθεί σωστά
! μπορεί όμως οι 2 πρώτοι να μην είναι ταξινομημένοι ακόμη
Αν α > β τότε Αντιμετάθεσε α, β ! ταξινόμησε τους 2 πρώτους

Εμφάνισε α, β, γ

Τέλος
```

Άσκηση 77 - αναζήτηση σε δισδιάστατο πίνακα

Για την 5ήμερη εκδρομή της Γ Λυκείου ενός σχολείου, οι 60 μαθητές της, αποφάσισαν να διεξάγουν μια λαχειοφόρο αγορά, πουλώντας λαχνούς. Κάθε μαθητής πούλησε από 30 λαχνούς. Σε έναν πίνακα `ΑΡΙΘ[60,30]` καταγράφτηκαν οι αριθμοί από τους λαχνούς των 60 μαθητών και σε ένα πίνακα `ΟΝ[60]` τα ονόματα των 60 μαθητών.

Να γραφεί αλγόριθμος σε ψευδογλώσσα, που με δεδομένο τον πίνακα `ΑΡΙΘ[]` και τον πίνακα `ΟΝ[]`, θα διαβάζει τον τυχερό αριθμό που κληρώθηκε και θα εμφανίζει ποιος μαθητής πούλησε τον τυχερό λαχνό.

Απάντηση

Θα σαρώσουμε τον `ΑΡΙΘ[]` γραμμή-γραμμή. Μας ενδιαφέρει η γραμμή στην οποία θα βρεθεί ο τυχερός αριθμός.

```
Αλγόριθμος Λαχνός
Δεδομένα // ΑΡΙΘ[60, 30], ΟΝ[60] //

Εμφάνισε "Δώστε τον τυχερό αριθμό : "
Διάβασε τυχερόςΑριθμός

κ ← 0
λ ← 0
βρέθηκε ← Ψευδής
Όσο όχι βρέθηκε και κ < 60 επανάλαβε
  κ ← κ + 1
  Όσο όχι βρέθηκε και λ < 30 επανάλαβε
    λ ← λ + 1
    Αν ΑΡΙΘ[κ, λ] = τυχερόςΑριθμός τότε
      βρέθηκε ← Αληθής
```

```
      δείκτηςΜαθ ← κ
    Τέλος_αν
  Τέλος_επανάληψης
Τέλος_επανάληψης

Εμφάνισε "Ο μαθητής που το πούλησε είναι : ", ΟΝ[δείκτηςΜαθ]
Τέλος
```

Άσκηση 78 - αντιμετάθεση μονών-ζυγών χωρίς σημασία στην σειρά

Ένας μονοδιάστατος πίνακας ακεραίων **A[N]**, όπου **N** άρτιος, περιέχει όλους τους φυσικούς αριθμούς από το 1 μέχρι το Ν ακριβώς μία φορά (ανακατεμένους). Επειδή το Ν είναι άρτιος αριθμός, ο πίνακας περιέχει ίσο πλήθος μονών και ζυγών αριθμών. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να αντιμεταθέτει τους μονούς με τους ζυγούς ακεραίους. Οι θέσεις των αριθμών δεν έχουν καμία σημασία. Αρκεί στο τέλος του αλγορίθμου να έχει τοποθετηθεί μονός αριθμός εκεί που πριν υπήρχε ζυγός και αντίστροφα.

Απάντηση

Αφού η σειρά των αριθμών δεν έχει καμία σημασία τότε αρκεί σε κάθε θέση να θυμόμαστε ποιον μονό ή ζυγό τοποθετήσαμε τελευταίο.

```
Αλγόριθμος ΑντιμετάθεσηΜονώνΖυγών
Ν ← 6
Α[1] ← 4
Α[2] ← 2
Α[3] ← 1
Α[4] ← 5
Α[5] ← 3
Α[6] ← 6

! κυρίως αλγόριθμος
τελευταίοςΜονός ← 1
τελευταίοςΖυγός ← 2
Για κ από 1 μέχρι Ν
  Αν Α[κ] mod 2 = 1 τότε                    ! αν είναι μονός
    Α[κ] ← τελευταίοςΖυγός
    τελευταίοςΖυγός ← τελευταίοςΖυγός + 2
  αλλιώς                                    ! αν είναι ζυγός
    Α[κ] ← τελευταίοςΜονός
    τελευταίοςΜονός ← τελευταίοςΜονός + 2
  Τέλος_αν
Τέλος_επανάληψης

Για κ από 1 μέχρι Ν
  Εμφάνισε Α[κ], "  "
Τέλος_επανάληψης
Τέλος
```

**** Άσκηση 79 - αντιμετάθεση μονών-ζυγών με την σειρά που εμφανίζονται***

Ίδια με την προηγούμενη άσκηση, αλλά τώρα η σειρά εμφάνισης έχει σημασία. Ο πίνακας περιέχει και πάλι άρτιο πλήθος συνεχόμενων φυσικών αριθμών από το 1 έως το Ν, ανακατεμένους. Θέλουμε να αντιμεταθέσουμε τους μονούς με τους ζυγούς, αλλά με την ίδια σειρά εμφάνισης. Δηλαδή ο πρώτος μονός του πίνακα να αντιμετατεθεί με τον πρώτο ζυγό, ο δεύτερος μονός με τον δεύτερο ζυγό κτλ.

Υπόδειξη: Αν χρησιμοποιήστε δύο δείκτες και εξωτερικό, δικό σας, πίνακα η άσκηση δεν είναι δύσκολη.

Απάντηση

Θα χρησιμοποιήσουμε δύο δείκτες, **μονός**, **ζυγός**. Και οι δύο δείκτες θα ξεκινούν από αριστερά και θα σαρώνουν τον πίνακα προς τα δεξιά. Ο ένας δείκτης θα σταματά σε κάθε μονό αριθμό και ο άλλος σε κάθε ζυγό. Για την μετακίνηση και το σταμάτημα των δεικτών θα εφαρμόσουμε την διαδικασία προσπέρασης.
Στα σημεία που σταμάτησαν οι δείκτες θα πρέπει να κάνουμε αντιμετάθεση. Όμως αυτό θα χαλάσει την σειρά μονών-ζυγών στον πίνακα και θα μπερδέψει τις επόμενες αναζητήσεις των δεικτών. Έτσι η αντιμετάθεση δεν θα γίνει επιτόπου αλλά σε νέο, δικό μας πίνακα **B[N]**. Όλα τα παραπάνω θα επαναληφθούν Ν/2 φορές.

```
Αλγόριθμος ΑντιμετάθεσηΜονώνΖυγών2
Ν ← 6
Α[1] ← 4
Α[2] ← 3
Α[3] ← 1
Α[4] ← 5
Α[5] ← 2
Α[6] ← 6

μονός ← 1    ! αρχικά και οι δύο δείκτες ..
ζυγός ← 1    ! ..δείχνουν την πρώτη θέση του πίνακα
Για κ από 1 μέχρι Ν div 2   ! τις μισές φορές απ' όσο είναι το Ν
  ! διαδικασία προσπέρασης για το μονός (προσπερνά όλα τα ζυγά)
  Όσο μονός < Ν και Α[μονός] mod 2 = 0 επανάλαβε
    μονός ← μονός + 1
  Τέλος_επανάληψης

  ! διαδικασία προσπέρασης για το ζυγός (προσπερνά όλα τα μονά)
  Όσο ζυγός < Ν και Α[ζυγός] mod 2 = 1 επανάλαβε
    ζυγός ← ζυγός + 1
  Τέλος_επανάληψης

  Β[μονός] ← Α[ζυγός]   ! χιαστί αντιμετάθεση
  Β[ζυγός] ← Α[μονός]
  μονός ← μονός + 1     ! βγάζουμε και τα δύο από τις θέσεις που είχαν ..
  ζυγός ← ζυγός + 1     ! ..σταματήσει ώστε να συνεχίσουν την αναζήτηση..
Τέλος_επανάληψης        ! ..στην επόμενη επανάληψη

Για κ από 1 μέχρι Ν
  Εμφάνισε Β[κ], "  "
Τέλος_επανάληψης
Τέλος
```

Η διαδικασία προσπέρασης είναι ένα δυνατό εργαλείο που μπορεί να μας ξελασπώσει σε προβλήματα που αρχικά φαντάζουν ακατόρθωτα.

Άσκηση 80 - είναι αριθμός;

Ένας πίνακας **A[N]** περιέχει χαρακτήρες, αλλά έναν μόνο χαρακτήρα σε κάθε θέση. Να γραφεί αλγόριθμος σε ψευδογλώσσα ο οποίος να ελέγχει αν ο πίνακας μπορεί να εκφράζει φυσικό αριθμό (δηλαδή όλες οι θέσεις του να είναι κάποιος θετικός ακέραιος σε μορφή χαρακτήρα). Το αποτέλεσμα του ελέγχου να καταχωρείται σε μια λογική μεταβλητή **είναιΑριθμός**. Για παράδειγμα αν **A=['3','2','5','4']** τότε **είναιΑριθμός=Αληθής**.

Οι χαρακτήρες αναπαριστώνται εσωτερικά στον υπολογιστή ως φυσικοί αριθμοί (όπως και όλα τα δεδομένα). Για να αντιστοιχίσει ο υπολογιστής τον αριθμό με τον χαρακτήρα χρησιμοποιεί τον διάσημο πίνακα ASCII. Η σύγκριση των αλφαριθμητικών γίνεται με βάση αυτόν τον πίνακα. Έτσι ένας χαρακτήρας χ *είναι μεγαλύτερος από έναν άλλον* ψ *αν αντιστοιχεί σε μεγαλύτερο αριθμό στον πίνακα ASCII.*
Τις περισσότερες φορές ο προγραμματιστής δεν χρειάζεται να γνωρίζει λεπτομέρειες για τον πίνακα ASCII. Πρέπει όμως να γνωρίζει ότι πρώτα τοποθετούνται στον πίνακα τα αριθμητικά ψηφία από το `'0'` *έως το* `'9'` *σε διπλανές θέσεις, ακολουθούν τα κεφαλαία γράμματα από το* `'Α'` *έως το* `'Ω'`*, επίσης σε διπλανές θέσεις και τέλος περιλαμβάνονται τα πεζά* `'α'` *έως* `'ω'` *και πάλι σε διπλανές θέσεις. Έτσι όταν συγκρίνουμε χαρακτήρες, οι αριθμοί είναι πάντα μικρότεροι από τα κεφαλαία και τα κεφαλαία μικρότερα από τα πεζά. Η σύγκριση αριθμών ή γραμμάτων μεταξύ τους ακολουθεί την φυσική διάταξη, για παράδειγμα* `'γ'<'ζ'`*, και* `'1'<'5'`*.*
Επομένως η διάταξη ASCII είναι τέτοια που μας επιτρέπει ελέγχους του τύπου `'0'≤χ≤'9'`*. Αν ισχύει η προηγούμενη ανισότητα τότε το* χ *περιέχει αριθμητικό ψηφίο. Επίσης αν* `'Α'≤χ≤'Ω'` *τότε το χ περιέχει κεφαλαίο γράμμα ενώ αν* `'α'≤χ≤'ω'` *τότε περιέχει πεζό.*

Απάντηση

Ο κλασικός αλγόριθμος είναι να ψάξουμε αν υπάρχει στοιχείο του πίνακα που να μην είναι αριθμός

```
Αλγόριθμος ΕίναιΑριθμόςΑλγ
Ν ← 4
Α[1] ← '2'
Α[2] ← '4'
Α[3] ← '1'
Α[4] ← '0'

είναιΑριθμός ← Αληθής
κ ← 0
Όσο είναιΑριθμός και κ < Ν επανάλαβε
  κ ← κ + 1
  είναιΑριθμός ←  Α[κ] ≥ '0' και Α[κ] ≤ '9' ! θα μπορούσαμε και με Αν...
Τέλος_επανάληψης
Εμφάνισε είναιΑριθμός
Τέλος
```

Αφού μάθαμε την διαδικασία προσπέρασης, μπορούμε να την χρησιμοποιήσουμε σε πάρα πολλά προβλήματα. Βεβαιωθείτε πρώτα ότι καταλάβατε πλήρως την διαδικασία προσπέρασης στην άσκηση 73. Δείτε τώρα μια λύση της άσκησης με διαδικασία προσπέρασης

```
...
κ ← 1
Όσο κ ≤ Ν και Α[κ] ≥ '0' και Α[κ] ≤ '9' επανάλαβε ! προσπερνά όλα όσα..
  κ ← κ + 1                                       ! ..ταιριάζουν
Τέλος_επανάληψης
είναιΑριθμός ← κ = Ν+1
...
```

Πως δουλεύει η προσπέραση; Ξεκινώντας από την αρχή του πίνακα, προσπερνάμε όλους τους χαρακτήρες που ταιριάζουν στην συνθήκη (εδώ, όλους όσους είναι

αριθμοί). Αν πάνε όλα καλά, τότε η προσπέραση θα έχει πάει τον δείκτη έξω από τον πίνακα. Αν κάτι πάει στραβά, τότε ο δείκτης θα μείνει μέσα στον πίνακα. Δείτε και την άσκηση 81.

Άσκηση 81 - τομή συνόλων (η διαδικασία προσπέρασης ως αναζήτηση)

Έχουμε δύο πίνακες χαρακτήρων **A[N]** και **B[M]** οι οποίοι περιέχουν έναν μόνο χαρακτήρα σε κάθε θέση. Οι πίνακες αναπαριστούν σύνολα, δηλαδή δεν περιέχουν διπλότυπα. Θέλουμε τα κοινά στοιχεία των δύο πινάκων, δηλαδή στοιχεία τα οποία να εμφανίζονται και στους δύο πίνακες (την τομή τους).

(α) Γράψτε έναν αλγόριθμο σε ψευδογλώσσα που να βρίσκει και να εμφανίζει στην οθόνη τα κοινά στοιχεία.

* (β) Πως θα αντιμετωπίσουμε το ίδιο πρόβλημα αν γνωρίζουμε ότι οι δύο πίνακες είναι ταξινομημένοι;

Απάντηση

(α) Θα ψάξουμε ένα- ένα τα στοιχεία του ενός πίνακα μέσα στον άλλον. Ο παραδοσιακός τρόπος να γράψουμε τον κώδικα είναι ο παρακάτω

```
Αλγόριθμος ΤομήΣυνόλων
N ← 5
A[1] ← 'α'
A[2] ← 'φ'
A[3] ← 'ε'
A[4] ← 'ζ'
A[5] ← 'β'

M ← 3
B[1] ← 'ε'
B[2] ← 'ψ'
B[3] ← 'ζ'

Για κ από 1 μέχρι N                      ! το κ σαρώνει τον A
  βρέθηκε ← Ψευδής
  λ ← 1                                  ! το λ σαρώνει τον B
  Όσο όχι βρέθηκε και λ ≤ M επανάλαβε
    Αν A[κ] = B[λ] τότε βρέθηκε ← Αληθής ! ή: βρέθηκε ← A[κ] = B[λ]
    λ ← λ + 1
  Τέλος_επανάληψης
  Αν βρέθηκε τότε Εμφάνισε A[κ]
Τέλος_επανάληψης
Τέλος
```

Μπορούμε να δώσουμε στον προηγούμενο κώδικα μια πιο φρέσκια όψη χρησιμοποιώντας την διαδικασία προσπέρασης ως μηχανισμό αναζήτησης. Η ιδέα είναι να προσπερνάμε τους χαρακτήρες που δεν ταιριάζουν μέχρι να βρούμε το ζητούμενο. Αν το βρούμε, τότε έχουμε και τον δείκτη του. Αν δεν το βρούμε τότε ο δείκτης θα έχει ξεπεράσει κατά 1 τη διάσταση του πίνακα. Φυσικά δεν αλλάζει η απόδοση του κώδικα. Η αναζήτηση είναι πάλι σειριακή, αλλά ο κώδικας γίνεται συντομότερος (και πιο μοντέρνος)

```
...
Για κ από 1 μέχρι N                    ! το κ σαρώνει τον A
  λ ← 1                                ! το λ σαρώνει τον B
  Όσο λ ≤ M και A[κ] ≠ B[λ] επανάλαβε ! προσπέρνα όποιο δεν ταιριάζει
    λ ← λ + 1
  Τέλος_επανάληψης
  Αν λ < M + 1 τότε Εμφάνισε A[κ]     ! αν το λ δεν βγήκε έξω από τον..
```

```
Τέλος_επανάληψης                    ! ..πίνακα, τότε το στοιχείο βρέθηκε
...
```

* (β) Αν οι πίνακες είναι ταξινομημένοι, τότε ο αλγόριθμος μπορεί να επιταχυνθεί σημαντικά. Κάθε φορά που ψάχνουμε ένα στοιχείο του πίνακα `A[]` στον `B[]`, δεν χρειάζεται να φτάνουμε μέχρι το τέλος του `B`. Σταματάμε την αναζήτηση μόλις τα στοιχεία του `B` γίνουν μεγαλύτερα από το στοιχείο που ψάχνουμε. Στη συνέχεια, η αναζήτηση του επόμενου στοιχείου του `A` θα ξεκινήσει από 'κει που σταμάτησε η αναζήτηση του προηγούμενου. Έτσι δεν θα χρειαστεί να σαρώσουμε τον πίνακα `B` πολλές φορές, αλλά μόνο μία. Προσέξτε ότι το λ αρχικοποιείται μία μόνο φορά και αυξάνει συνεχώς, χωρίς πισωγυρίσματα.

```
Αλγόριθμος ΤομήΣυνόλων2
Ν ← 5
Α[1] ← 'α'
Α[2] ← 'β'
Α[3] ← 'ε'
Α[4] ← 'ζ'
Α[5] ← 'φ'

Μ ← 3
Β[1] ← 'ε'
Β[2] ← 'ζ'
Β[3] ← 'ψ'

λ ← 1                               ! το λ σαρώνει τον Β
Για κ από 1 μέχρι Ν                 ! το κ σαρώνει τον Α
  Όσο λ ≤ Μ και Α[κ] > Β[λ] επανάλαβε ! προσπέρνα όλα τα μικρότερα Β
    λ ← λ + 1
  Τέλος_επανάληψης                  ! η προσπέραση τελειώνει εκεί που..
                                    ! ..το Α είναι μικρότερο-ίσο του Β
  Αν λ < Μ + 1 και Β[λ] = Α[κ] τότε   ! αν είναι ίσο, το βρήκαμε..
    Εμφάνισε Α[κ]
  Τέλος_αν                          ! ..αλλιώς, πάμε στο επόμενο Α
Τέλος_επανάληψης
Τέλος
```

Άσκηση 82 - καρκινικές φράσεις με προσπέραση

Το ίδιο με την άσκηση 43, αλλά τώρα προσπαθήστε να την λύσετε με διαδικασία προσπέρασης. Να θυμηθούμε σύντομα ότι θέλουμε να ελέγξουμε αν μία φράση είναι καρκινική ή όχι.

Απάντηση

Η διαδικασία προσπέρασης δίνει λιτό και κομψό κώδικα, όπου μπορεί να εφαρμοστεί. Η ιδέα είναι να προσπεράσουμε όσα στοιχεία “ταιριάζουν”. Αν τα προσπεράσουμε όλα, τότε θα έχουμε βγει έξω από το διάστημα της επανάληψης. Για να προλάβουμε έναν κουτό διερμηνευτή, απ' το να κάνει την ίδια πράξη ξανά και ξανά, κάνουμε την πράξη μία φορά και την καταχωρούμε σε μια νέα μεταβλητή `κέντρο`.

```
κέντρο ←  Ν div 2
κ ← 1
Όσο κ ≤ κέντρο και Α[κ] = Α[Ν+1 - κ] επανάλαβε
  κ ← κ + 1
Τέλος_επανάληψης
είναιΚαρκ κ = κέντρο + 1
```

Άσκηση 83 - πετώντας γάτες απ' το μπαλκόνι (σειριακή και δυαδική αναζήτηση)

Ένα πολυόροφο κτίριο έχει `N` ορόφους. Μία γάτα πεθαίνει αν πέσει από τον `M` όροφο και πάνω αλλά ζει αν ο όροφος από τον οποίο πέσει είναι μικρότερος του `M`. Μια συνάρτηση `Έζησε(ν)` δέχεται τον αριθμό του ορόφου και επιστρέφει την τιμή `Αληθής` αν η γάτα έζησε και `Ψευδής` αν πέθανε. Διαθέτουμε άπειρες γάτες για δοκιμές και θέλουμε να βρούμε από ποιον όροφο `M` και πάνω πεθαίνουν οι γάτες. Να θεωρήσετε ότι υπάρχει οπωσδήποτε κάποιος όροφος `M`, με `1 ≤ M ≤ N`, πάνω από τον οποίο πεθαίνουν οι γάτες.

(α) Να γράψετε αλγόριθμο σε ψευδογλώσσα που να βρίσκει το `M` και να είναι όσο το δυνατόν φιλικός προς τις γάτες, δηλαδή να θυσιάσουμε όσο το δυνατόν λιγότερες γάτες.

(β) Το ίδιο αλλά τώρα θέλουμε τον πιο μοχθηρό αλγόριθμο για τις γάτες.

* (γ) Προσπαθήστε να βρείτε μια πολιτική η οποία να είναι όσο το δυνατόν φιλικότερη προς τον άνθρωπο που κάνει τις δοκιμές. Θέλουμε να κάνουμε τις λιγότερες δυνατόν δοκιμές ώστε να μην κουραστεί ο άνθρωπος, χωρίς να μας ενδιαφέρει πόσες γάτες θα πεθάνουν. Γράψτε τον αλγόριθμο μόνο σε φυσική γλώσσα και μετά, αν μπορείτε, προσπαθήστε να τον μετατρέψετε σε ψευδογλώσσα.

Απάντηση

(α) Για να θυσιάσουμε τις λιγότερες δυνατόν γάτες θα ξεκινήσουμε τις δοκιμές από κάτω προς τα πάνω. Μόλις πεθάνει η πρώτη γάτα, θα έχουμε βρει τον όροφο.

```
Αλγόριθμος Φιλικός
Δεδομένα // N //

ν ← 1
Όσο Έζησε(ν) επανάλαβε
  ν ← ν + 1
Τέλος_Επανάληψης

Αποτελέσματα // ν //
Τέλος
```

(β) Το ίδιο με το προηγούμενο, αλλά τώρα θα ξεκινήσουμε από πάνω προς τα κάτω. Προσέξτε ότι τώρα πρέπει να προσθέσουμε 1 στον όροφο που θα ζήσει η πρώτη γάτα (η άσκηση ζητά τον μικρότερο όροφο `M` στον οποίο πεθαίνει η γάτα)

```
Αλγόριθμος Μοχθηρός
Δεδομένα // N //

ν ← N
Όσο Όχι Έζησε(ν) επανάλαβε
  ν ← ν - 1
Τέλος_Επανάληψης

Αποτελέσματα // ν+1 //
Τέλος
```

* (γ) Δεν είναι δύσκολο να βρούμε την πολιτική με τις λιγότερες δοκιμές. Πρώτα θα δοκιμάσουμε ακριβώς στον μεσαίο όροφο. Αν η γάτα πεθάνει θα περιοριστούμε στο κάτω μισό, αλλιώς στο πάνω μισό. Αφού εντοπίσουμε σε ποιο μισό είναι ο όροφος που ψάχνουμε θα ξαναδοκιμάσουμε στο μισό του μισού. Αυτό θα συνεχιστεί μέχρι να μείνουν μόνο δύο διπλανοί όροφοι. Τότε, μπορεί να χρειαστεί μία ακόμη δοκιμή για να εντοπίσουμε τον σωστό όροφο.

Βλέπετε, η εύρεση της έξυπνης πολιτικής και η περιγραφή της σε φυσική γλώσσα είναι πολύ εύκολη. Όταν ψάχνουμε κάτι, μειώνοντας συνεχώς το διάστημα αναζήτησης στο μισό, τότε λέμε ότι κάνουμε *δυαδική αναζήτηση*. Η μετατροπή βέβαια της πολιτικής σε κώδικα είναι λίγο πιο δύσκολη, αλλά όχι ακατόρθωτη. Πρώτον, πρέπει να ξέρουμε σε κάθε βήμα, το διάστημα των ορόφων μέσα στο οποίο συνεχίζεται η αναζήτηση. Για τον σκοπό αυτό θα χρησιμοποιήσουμε δύο δείκτες, **πάνω** και **κάτω**, οι οποίοι θα δείχνουν τα όρια των ορόφων μέσα στους οποίους συνεχίζουμε να ψάχνουμε. Ο μεσαίος όροφος στον οποίο θα κάνουμε τις δοκιμές είναι ασφαλώς ο μέσος όρος τους. Ο κώδικάς πρέπει να έχει μία δομή επανάληψης, η οποία να σταματά όταν οι δείκτες **πάνω** και **κάτω** δείξουν σε διπλανές θέσεις. Μέσα στο σώμα της επανάληψης θα γράψουμε με εντολές αυτό που περιγράψαμε με φυσική γλώσσα. Αφού βγούμε από την δομή της επανάληψης, αρκεί μία τελευταία δοκιμή για να αποφασίσουμε ποιος είναι ο ζητούμενος όροφος. Με όλα αυτά κατά νου, δεν είναι δύσκολο να γράψουμε τον κώδικα. Δείτε πόσο λιτός είναι ο κώδικας σε σχέση με τα πολλά λόγια της φυσικής γλώσσας!

```
Αλγόριθμος Έξυπνος
Δεδομένα // N //

πάνω ← N
κάτω ← 1
Όσο πάνω > κάτω+1 επανάλαβε     ! μέχρι να έρθουν σε διπλανές θέσεις
  κέντρο ← (πάνω + κάτω) div 2  ! κέντρο είναι ο μέσος όρος τους
  Αν Έζησε(κέντρο) τότε
    κάτω ← κέντρο               ! το κάτω πρέπει να δείχνει εκεί που ζει
  αλλιώς
    πάνω ← κέντρο            ! το πάνω πρέπει να δείχνει εκεί που πεθαίνει
  Τέλος_Αν
Τέλος_Επανάληψης

ν ← 0
Αν Έζησε(κάτω) τότε
  ν ← πάνω
αλλιώς
  ν ← κάτω
Τέλος_Αν

Αποτελέσματα // ν //
Τέλος
```

Άσκηση 84 - δεύτερο μεγαλύτερο στοιχείο

Σε έναν μονοδιάστατο πίνακα πραγματικών αριθμών **A[N]** θέλουμε να βρούμε το δεύτερο μεγαλύτερο στοιχείο. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να το βρίσκει. Θεωρείστε ότι ο πίνακας έχει μέγεθος τουλάχιστον 2 και ότι το δεύτερο μεγαλύτερο στοιχείο υπάρχει (δεν είναι δηλαδή όλα ίσα μεταξύ τους)

Απάντηση

Ο αλγόριθμος είναι ίδιος με αυτόν της εύρεσης του μεγαλύτερου στοιχείου, μόνο που τώρα θα κρατάμε τα δύο μεγαλύτερα στοιχεία που έχουν βρεθεί. Για κάθε επόμενο στοιχείο θα βρίσκουμε το διάστημα στο οποίο ανήκει (χρησιμοποιώντας έξυπνα την **αλλιώς_αν**). Ο σκοπός είναι μετά από κάθε σύγκριση να είμαστε σίγουροι για το ποια είναι τα δύο μεγαλύτερα στοιχεία του πίνακα μέχρι εκείνη τη στιγμή. Αρχικά θα αποφασίσουμε για το ποιο είναι πρώτο και δεύτερο συγκρίνοντας

τα δύο πρώτα στοιχεία του πίνακα. Αν αυτά είναι ίσα τότε το πρώτο και το δεύτερο μπορούν να είναι το ίδιο στοιχείο.

```
Αν Α[1] > Α[2] τότε
  πρώτο ← Α[1]
  δεύτερο ← Α[2]
αλλιώς_αν Α[1] < Α[2] τότε
  πρώτο ← Α[2]
  δεύτερο ← Α[1]
αλλιώς
  πρώτο ← Α[1]
  δεύτερο ← Α[1]
Τέλος_αν

Για κ από 3 μέχρι Ν
  Αν Α[κ] > πρώτο τότε          ! είναι το πιο μεγάλο απ' όλα
    δεύτερο ← πρώτο             ! κατέβασε το πρώτο μία θέση κάτω
    πρώτο ← Α[κ]                ! κάνε το νέο στοιχείο πρώτο
  αλλιώς_αν Α[κ] > δεύτερο τότε ! είναι σίγουρα το δεύτερο μεγαλύτερο
    δεύτερο ← Α[κ]
  Τέλος_αν
Τέλος_επανάληψης
```

Η απλή δομή των δύο μεταβλητών **πρώτος**, **δεύτερος** που χρησιμοποιούμε εδώ είναι στην πραγματικότητα μια *ουρά προτεραιότητας*. Η ουρά αυτή είναι μια κανονική ουρά αλλά όταν έρχεται κάποιο νέο μέλος τότε έχει προτεραιότητα (σύμφωνα με κάποιο κριτήριο) και έτσι μπορεί να προχωρήσει μέσα στην ουρά αντί να μείνει τελευταίο. Στην περίπτωσή μας το κριτήριο είναι η αξία του αριθμού. Έτσι ο κάθε αριθμός προχωρά μέσα στην ουρά μέχρι να συναντήσει κάποιον μεγαλύτερο. Αφού μπει στη σωστή θέση μετακινεί όλα τα μικρότερα στοιχεία προς τα πίσω και πετά έξω από την ουρά αυτό που ήταν στην άκρη. Αυτό που υλοποιήσαμε σ' αυτή την άσκηση είναι μια τέτοια ουρά προτεραιότητας με δύο μέλη.

***** Άσκηση 85 - ν-μεγαλύτερο στοιχείο (ουρά προτεραιότητας)***

Σε έναν μονοδιάστατο πίνακα φυσικών αριθμών **Α[Ν]** θέλουμε να βρούμε το ν-μεγαλύτερο στοιχείο. Υποθέστε ότι ο πίνακας περιέχει θετικούς ακεραίους. Να γραφεί πρόγραμμα σε ΓΛΩΣΣΑ που να βρίσκει τον ν-μεγαλύτερο ακέραιο. Θεωρείστε ότι ο πίνακας έχει μέγεθος τουλάχιστον ν και ότι το ν-μεγαλύτερο στοιχείο υπάρχει (δεν είναι δηλαδή πολλά ίσα στοιχεία μεταξύ τους, ώστε να μην υπάρχει). Για το πρόγραμμα της ΓΛΩΣΣΑ κάντε πρακτική εφαρμογή για **Ν=10** και **ν=5**.

Απάντηση

Στο πλαίσιο του μαθήματος ΑΕΠΠ ο ενδεδειγμένος τρόπος να λύσουμε την άσκηση είναι με ταξινόμηση. Σε μικρούς πίνακες το κόστος είναι αποδεκτό. Αν όμως ο πίνακας είναι μεγάλος τότε είναι απαράδεκτο να ταξινομήσουμε ολόκληρο τον πίνακα για να βρούμε πχ το 5ο μεγαλύτερο στοιχείο. Στις περιπτώσεις αυτές χρησιμοποιούμε την ουρά προτεραιότητας. Ο αλγόριθμος είναι μια επέκταση του προηγούμενου, μόνο που τώρα πρέπει να γράψουμε μια χωριστή διαδικασία για την εισαγωγή στην ουρά. Χρειαζόμαστε επίσης και μια διαδικασία για την αρχικοποίηση της ουράς. Στην αρχικοποίηση απλά μηδενίζουμε όλα τα στοιχεία της ουράς.

```
! Μηδενίζει τον πίνακα που θα κρατά την ουρά
ΔΙΑΔΙΚΑΣΙΑ Αρχικοποίηση(Α)
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Α[5], κ
```

```
ΑΡΧΗ
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 5
    Α[κ] <- 0
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Η καρδιά του αλγόριθμου είναι η διαδικασία που εισάγει τα νέα στοιχεία στην ουρά. Η διαδικασία χωρίζεται σε τρία μέρη. Πρώτα βρίσκουμε την σωστή θέση του νέου στοιχείου (όσο μεγαλύτερο είναι το στοιχείο τόσο πιο μπροστά πρέπει να τοποθετηθεί). Έπειτα μετακινούμε όλα τα χαμηλότερα στοιχεία μία θέση πίσω για να αφήσουμε χώρο για το νέο στοιχείο (το στοιχείο που ήταν τελευταίο στην ουρά χάνεται). Τέλος τοποθετούμε το νέο στοιχείο στη σωστή θέση.

```
! Βάζει το χ στη σωστή θέση του πίνακα Α
ΔΙΑΔΙΚΑΣΙΑ Βάλε(Α, χ)
ΣΤΑΘΕΡΕΣ
  Ν = 5
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Α[Ν], χ, θέση, κ
ΑΡΧΗ
 ! 1 - βρες την σωστή θέση για το χ
  θέση <- 0
  ΟΣΟ θέση < Ν ΚΑΙ χ > Α[θέση+1] ΕΠΑΝΑΛΑΒΕ     ! σύντομη αποτίμηση
    θέση <- θέση + 1
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ! 2 – μετακίνησε όλα τα μικρότερα μία θέση πίσω
  ΑΝ θέση > 0 ΤΟΤΕ         ! Αν βρέθηκε θέση στο προηγούμενο βήμα
    ΓΙΑ κ ΑΠΟ 2 ΜΕΧΡΙ θέση
      Α[κ - 1] <- Α[κ]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

    ! 3 - τοποθέτησε το χ στη σωστή θέση
    Α[θέση] <- χ
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Τώρα πλέον το κυρίως πρόγραμμα δεν έχει παρά να αρχικοποιήσει την ουρά και μετά να εισάγει όλα τα στοιχεία ένα προς ένα μέσα στην ουρά. Στο τέλος η ουρά θα περιέχει τα 5 μεγαλύτερα στοιχεία, οπότε το τελευταίο στοιχείο της ουράς θα είναι το ζητούμενο. Παρατηρείστε ότι όλη η διαδικασία χρειάζεται μία μόνο σάρωση του πινάκα χωρίς να τον χαλάσουμε όπως θα κάναμε με την ταξινόμηση.

```
ΠΡΟΓΡΑΜΜΑ Ν_ΜΕΓΑΛΥΤΕΡΑ
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Ουρά[5], Α[10], κ
ΑΡΧΗ
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 10
    ΔΙΑΒΑΣΕ Α[κ]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΚΑΛΕΣΕ Αρχικοποίηση(Ουρά)
  ! βάλε όλα τα στοιχεία στην ουρά
  ΓΙΑ κ ΑΠΟ 1 ΜΕΧΡΙ 10
    ΚΑΛΕΣΕ Βάλε(Ουρά, Α[κ])
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

  ΓΡΑΨΕ “Το 5ο μεγαλύτερο στοιχείο είναι: “, Ουρά[1]
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Στην γραμμή `ΟΣΟ θέση<N ΚΑΙ χ>Α[θέση+1]` *χρησιμοποιούμε "σύντομη αποτίμηση". Αυτό σημαίνει ότι όταν* `θέση<N` *προκύψει ψευδής, τότε δεν χρειάζεται να ελέγξουμε την συνθήκη μετά το* `ΚΑΙ`*, επειδή έτσι κι αλλιώς όλη η έκφραση είναι ψευδής. Κάθε φορά που θα αποτυγχάνει το* `θέση<Ν`*, το δεύτερο μέρος* `χ>Α[θέση+1]` *δεν θα ελέγχεται. Έτσι ποτέ δεν θα προκληθεί λάθος επειδή προσπαθούμε να προσπελάσουμε μνήμη έξω από τον πίνακα.*
*Δεν υπάρχει στα ελληνικά επίσημη ορολογία για την "σύντομη αποτίμηση". Στα αγγλικά ονομάζεται "αποτίμηση βραχυκύκλωμα" (**short-circuit evaluation**). Άλλες ονομασίες στα ελληνικά είναι μερική αποτίμηση ή πρόωρη αποτίμηση (και τα δύο όμως έχουν μία αρνητική έννοια στην ελληνική γλώσσα).*
*Η σύντομη αποτίμηση δεν είναι κάτι εξαιρετικό ή προχωρημένο. Είναι ο συνηθισμένος τρόπος που υπολογίζει τις λογικές εκφράσεις ο ανθρώπινος εγκέφαλος. Πάρτε για παράδειγμα την λογική πρόταση: **`ΑΝ`** το τυρί μυρίζει άσχημα **`Η`** έχει αλλάξει χρώμα **`ΤΟΤΕ`** είναι χαλασμένο. Μπορούμε να χρησιμοποιήσουμε έναν τυφλό και μια σειρά από τυριά που μυρίζουν άσχημα και πάντα ο τυφλός θα διακρίνει αν το τυρί είναι χαλασμένο. Όταν του δώσουμε ένα τυρί που δεν μυρίζει άσχημα, τότε μόνο δεν θα μπορέσει να δώσει απάντηση. (Στην πραγματικότητα ο ανθρώπινος εγκέφαλος δουλεύει ακόμη πιο έξυπνα. Ο τυφλός θα μπορούσε να δώσει σωστή απάντηση ακόμη κι αν του δίναμε την λογική πρόταση με την συνθήκη αντεστραμμένη.)*
Το αντίθετο της σύντομης αποτίμησης είναι η πλήρης αποτίμηση των λογικών εκφράσεων. Αυτό σημαίνει ότι ο διερμηνευτής (ή μεταφραστής) της γλώσσας θα υπολογίσει ολόκληρη την έκφραση πριν καταλήξει στο τελικό αποτέλεσμα. Στην περίπτωση αυτή η εντολή `ΟΣΟ θέση<N ΚΑΙ χ>Α[θέση+1]` *θα προκαλέσει σφάλμα μόλις το* `θέση=Ν`*.*
Η σύντομη αποτίμηση είναι σήμερα ο μόνος αποδεκτός τρόπος υπολογισμού των λογικών εκφράσεων. Ο κώδικας είναι ταχύτερος, μικρότερος σε μέγεθος, πιο ευανάγνωστος και το σημαντικότερο πιο κοντά στην ανθρώπινη λογική. Τα πράγματα όμως δεν ήταν πάντα έτσι.
Η κόντρα μεταξύ πλήρους και σύντομης αποτίμησης έχει μεγάλη ιστορία. Στις πρωτόγονες εκδόσεις των γλωσσών BASIC και PASCAL (στα μέσα της δεκαετίας του 70) εφαρμόστηκε πλήρης αποτίμηση, επειδή ήταν ευκολότερη στην υλοποίηση και κανείς δεν έδινε μεγάλη σημασία σε μια τέτοια λεπτομέρεια. Τεχνικά και προγραμματιστικά το θέμα λύθηκε πολύ σύντομα, αλλά το κακό είχε ήδη γίνει. Είχαν γραφεί προγράμματα για μεγάλες εταιρείες, τα οποία στηριζόταν στην πλήρη αποτίμηση. Το πρόβλημα ήταν ότι κάποιοι προγραμματιστές μπορεί στο δεύτερο μέρος της λογικής έκφρασης να περιλάμβαναν κλήσεις συναρτήσεων με παράπλευρα αποτελέσματα. Αν άλλαζε ο τρόπος υπολογισμού των εκφράσεων τότε αυτές οι συναρτήσεις δεν θα καλούνταν ποτέ. Έτσι τα προγράμματα δεν θα ήταν συμβατά με τις νεότερες εκδόσεις των γλωσσών. Για τον λόγο αυτό η πλήρης αποτίμηση παρέμεινε στις BASIC και PASCAL για λόγους συμβατότητας με το παρελθόν.

Από τις αρχές της δεκαετίας του 80 η πλήρης αποτίμηση δέχτηκε σκληρή κριτική, κυρίως μέσα από την ακαδημαϊκή κοινότητα και το αντίπαλο στρατόπεδο της C (η C υποστήριξε σύντομη αποτίμηση από την πρώτη στιγμή). Τότε το υποδειγματικό μάρκετινγκ της Microsoft (με αιχμή του δόρατος τις GWBasic και quickBasic) και της θρυλικής Borland (με την Turbo Pascal) βαφτίσανε την ατέλεια των γλωσσών τους "χαρακτηριστικό" που δεν μπερδεύει τους αρχάριους προγραμματιστές. Έτσι έπεισαν τους πάντες ότι η πλήρης αποτίμηση είναι καλύτερη. Την δεκαετία του 90 η Borland χαλάρωσε την πολιτική της, αφήνοντας όσους θέλανε να μεταγλωττίζουν τον κώδικα με σύντομη αποτίμηση, μέχρι να χαθεί η εταιρεία και η PASCAL. Η Microsoft επέμενε στην πλήρη αποτίμηση (με την Visual Basic) μέχρι το 2002. Από την έκδοση .NET και μετά εισήγαγε δύο νέους τελεστές (`andAlso, orElse`*) για την σύντομη αποτίμηση.*

Η ιστορία θα ήταν απλά διασκεδαστική αν δεν είχε επιπτώσεις μέχρι και σήμερα. Το πρόβλημα είναι ότι πολλοί σημερινοί καθηγητές, που σπούδασαν την δεκαετία του 80 και 90, διδάχθηκαν την πλήρη αποτίμηση ως δόγμα (τη C τότε την χρησιμοποιούσε μόνο η ελίτ και η Java δεν υπήρχε ακόμη). Ανάμεσά τους βρίσκει κανείς ακόμη και σήμερα ένθερμους υποστηρικτές της πλήρους αποτίμησης. Το ποσοστό δεν πρέπει να είναι μεγάλο, αλλά σίγουρα υπάρχει. Πολλοί απ' αυτούς είναι βαθμολογητές στις εξετάσεις.

Έτσι κινδυνεύει ένα άψογος και κομψός αλγόριθμος να πέσει σε εχθρικό βαθμολογητή και να θεωρηθεί λάθος. Υπάρχουν δύο πράγματα που μπορείτε να κάνετε για να είστε καλυμμένοι:

(α) Το βιβλίο δεν αναφέρεται καθόλου στο θέμα. Παρά το ότι αφιερώνει μία σελίδα στο GOTO, δεν βρίσκει μία παράγραφο για την σύντομη αποτίμηση. Έτσι από πλευράς θεωρίας είστε καλυμμένοι (στην ψευδογλώσσα ακόμη ισχυρότερα). Αυτό που πρέπει να κάνετε είναι μέσα στο κείμενο αλλά και με σχόλιο στην αντίστοιχη γραμμή του κώδικα να τονίσετε ότι χρησιμοποιείται σύντομη αποτίμηση.

(β) Αν η προηγούμενη μέθοδος δεν είναι αρκετά πειστική και θέλετε να αποκλείσετε και την παραμικρή πιθανότητα να χάσετε μονάδες, τότε δεν πρέπει να χρησιμοποιήσετε σύντομη αποτίμηση. Υπάρχουν δύο τεχνικές για να την αποφύγετε:

(i) Η παραδοσιακή λύση είναι να κόψετε την δεύτερη συνθήκη από την κεφαλή της επανάληψης (ή της `Αν`*) και να την μετακινήσετε μέσα στο σώμα, κάπως έτσι*

```
ΟΣΟ θέση < Ν ΕΠΑΝΑΛΑΒΕ
  ΑΝ χ > Α[θέση+1] ΤΟΤΕ
    θέση <- θέση + 1
  ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Ο κώδικας βέβαια που προκύπτει είναι πολύ άσχημος. Αν εφαρμόζαμε την τεχνική αυτή σε `Αν`*, θα καταλήγαμε σε φωλιασμένα* `Αν` *(τα οποία απαγορεύονται).*

(ii) Ένας πιο κομψός τρόπος είναι να σταματήσετε την επανάληψη πριν το τελευταίο στοιχείο του πίνακα. Μετά επαναλαμβάνετε το σώμα της επανάληψης και για το τελευταίο στοιχείο, κάπως έτσι

```
ΟΣΟ θέση < Ν-1 ΚΑΙ χ > Α[θέση+1] ΕΠΑΝΑΛΑΒΕ  ! μέχρι το Ν-1
  θέση <- θέση + 1
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΑΝ χ > Α[Ν] ΤΟΤΕ     ! για το τελευταίο στοιχείο χωριστός έλεγχος
  θέση <- θέση + 1
ΤΕΛΟΣ_ΑΝ
```

Άσκηση 86 - έλεγχος ταξινόμησης

Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ελέγχει αν ένας πίνακας `Α[Ν]` είναι ταξινομημένος ή όχι

Απάντηση

Αν βρούμε έστω κι ένα στοιχείο του πίνακα σε λάθος θέση, τότε ο πίνακας δεν είναι ταξινομημένος. Για την υλοποίηση θα χρησιμοποιήσουμε την *διαδικασία προσπέρασης* (δείτε και άσκηση 73), μόνο που τώρα θα προσπερνάμε όσα στοιχεία βρίσκουμε ταξινομημένα. Θα χρησιμοποιήσουμε σύντομη αποτίμηση για ένα κομψό αποτέλεσμα

```
κ ← 1
Όσο κ < Ν και Α[κ] < Α[κ+1] επανάλαβε  ! σύντομη αποτίμηση
  κ ← κ + 1
Τέλος_Επανάληψης
είναιΤαξινομημένος ← κ = Ν    ! αν το κ έχει φτάσει το Ν, τότε όλα ΟΚ
```

Άσκηση 87 - ταξινόμηση με μέτρημα

Έχουμε έναν πίνακα ακεραίων `Α[Ν]` ο οποίος περιέχει αποκλειστικά και μόνο φυσικούς αριθμούς από το 1 έως το 100. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ταξινομεί τον πίνακα.

Απάντηση

Μπορούμε φυσικά να χρησιμοποιήσουμε έναν κλασικό αλγόριθμο ταξινόμησης. Όμως έτσι δεν θα εκμεταλλευτούμε το γεγονός ότι ο πίνακας περιέχει μόνο φυσικούς από το 1 μέχρι το 100. Η ιδέα είναι να μετρήσουμε πόσες φορές εμφανίζεται το 1 μέσα στον πίνακα, πόσες το 2,...κτλ. Αφού ξέρουμε πόσες φορές εμφανίζεται το καθένα, μπορούμε να ξαναγράψουμε τον πίνακα με την σωστή σειρά και το σωστό πλήθος εμφάνισης του κάθε αριθμού. Για το μέτρημα χρησιμοποιούμε έναν δικό μας εξωτερικό πίνακα, 100 θέσεων.

```
! Α[Ν] ο πίνακας προς ταξινόμηση
! Μετρ[100] ο πίνακας με τους μετρητές..
! ..ένας μετρητής για κάθε δυνατή τιμή

Για κ από 1 μέχρι 100 ! αρχικοποίηση του Μετρ[]
  Μετρ[κ] ← 0
Τέλος_επανάληψης

! το μέτρημα
Για κ από 1 μέχρι Ν
  Μετρ[Α[κ]] ← Μετρ[Α[κ]] + 1
Τέλος_επανάληψης

! ξαναγράφουμε τον πίνακα Α ταξινομημένο
δείκτης ← 1
```

```
Για κ από 1 μέχρι 100       ! κ = η τιμή που πρέπει να γράψουμε στον πίνακα
  Για λ από 1 μέχρι Μετρ[κ] ! γράφεται τόσες φορές όσες είναι το Μετρ[κ]
    Α[δείκτης] ← κ
    δείκτης ← δείκτης + 1   ! πήγαινε στην επόμενη θέση, ανεξαρτήτως του κ
  Τέλος_επανάληψης
Τέλος_επανάληψης
```

Η ταξινόμηση με μέτρημα είναι ο πιο γρήγορος τρόπος ταξινόμησης όταν ο πίνακας περιέχει τιμές από ένα πεπερασμένο σύνολο. Είναι το μόνο είδος ταξινόμησης που επιτυγχάνεται με μία και μόνο σάρωση των δεδομένων! Το μόνο του μειονέκτημα είναι η επιπλέον μνήμη που χρειάζεται.

**** Άσκηση 88 - διπλότυπο (εξαντλητικό και με μέτρημα)***

Ένας μονοδιάστατος πίνακας **A[N]** περιέχει αποκλειστικά και μόνο ακέραιους αριθμούς από 1 έως **N**. Γράψτε αλγόριθμο σε ψευδογλώσσα που να ελέγχει αν υπάρχει έστω κι ένα διπλότυπο μέσα στον πίνακα (ο ίδιος ακέραιος περισσότερες από μία φορές).

Απάντηση

Υπάρχουν πάρα πολλοί τρόποι για να εντοπίσουμε διπλότυπο. Ο πιο χοντροκομμένος είναι να ταξινομήσουμε τον πίνακα και μετά να ελέγξουμε αν υπάρχουν διπλανά στοιχεία που να είναι ίδια. Λίγο πιο έξυπνο είναι να σταματάμε την ταξινόμηση μόλις βρούμε ένα ζευγάρι ίδιων στοιχείων. Μπορούμε για παράδειγμα να γράψουμε μια παραλλαγή του bublesort, η οποία να σταματά μόλις βρει διπλανά στοιχεία ίδια. Θα πρέπει βέβαια να πείσουμε τον εαυτό μας ότι η μετακίνηση στοιχείων μέσα στον πίνακα είναι απαραίτητη!

Ένας επίσης αφελής τρόπος (ο οποίος όμως δεν χρειάζεται μετακινήσεις στοιχείων) είναι να συγκρίνουμε όλα τα στοιχεία με όλα και αν βρούμε κάποια ισότητα, τότε ο πίνακας περιέχει διπλότυπο. Φυσικά θα πρέπει να αποφύγουμε τις συγκρίσεις του κάθε στοιχείου με τον εαυτό του, αλλά και τις διπλές συγκρίσεις (δείτε άσκηση 40 και 46).

```
Αλγόριθμος ΔιπλότυποΕξαντλητικός
Εμφάνισε "Δώστε το μέγεθος του πίνακα"
Διάβασε N
! εισαγωγή πίνακα
Για κ από 1 μέχρι N
  Διάβασε Α[κ]
Τέλος_επανάληψης

! ο κυρίως αλγόριθμος
διπλότυπο ← Ψευδής          ! σαν σημαία, αλλά δεν σταματά την επανάληψη
Για κ από 1 μέχρι N
  Για λ από κ+1 μέχρι N
    Αν Α[κ] = Α[λ] τότε διπλότυπο ← Αληθής ! (*ΟΧΙ*) διπλότυπο ← Α[κ]=Α[λ]
  Τέλος_επανάληψης
Τέλος_επανάληψης

Εμφάνισε διπλότυπο
Τέλος
```

Το μειονέκτημα τώρα είναι ότι ο αλγόριθμος συνεχίζει να ψάχνει διπλότυπο ακόμη και όταν το έχει βρει. Ακόμη πιο αποδοτικό είναι να αντικαταστήσουμε το **Για-από-μέχρι** με **Όσο** και στην συνθήκη του **Όσο** να ελέγχουμε αν βρέθηκε διπλότυπο.

```
! ο κυρίως αλγόριθμος
```

```
διπλότυπο ← Ψευδής
κ ← 1
Όσο όχι διπλότυπο και κ < Ν επανάλαβε
  λ ← κ + 1
  Όσο όχι διπλότυπο και λ ≤ Ν επανάλαβε
    Αν Α[κ] = Α[λ] τότε διπλότυπο ← Αληθής !ή: διπλότυπο ← Α[κ] = Α[λ]
    λ ← λ + 1
  Τέλος_επανάληψης
  κ ← κ + 1
Τέλος_επανάληψης
```

Ο προηγούμενος αλγόριθμος είναι γενικός για οποιοδήποτε πίνακα. Στο συγκεκριμένο πρόβλημα όμως ξέρουμε ότι στον πίνακα περιέχονται μόνο αριθμοί από **1** έως **Ν**. Αυτό μπορούμε να το εκμεταλλευτούμε ως εξής. Θα δημιουργήσουμε έναν δικό μας πίνακα **Σ[Ν]**, στον οποίο θα γράφουμε την συχνότητα εμφάνισης του κάθε αριθμού. Αν η συχνότητα είναι 1 για όλους τους αριθμούς τότε δεν υπάρχουν διπλότυπα. Αν υπάρχει έστω και μία συχνότητα 2 τότε έχουμε διπλότυπο. Η απόδοση βέβαια αυτού του αλγόριθμου εκτοξεύεται, αφού θα χρειαστεί το πολύ-πολύ μία σάρωση για να αποφασίσει αν υπάρχει διπλότυπο.

```
! αρχικοποίηση του πίνακα συχνοτήτων
Για κ από 1 μέχρι Ν
  Σ[κ] ← 0
Τέλος_επανάληψης

! κυρίως αλγόριθμος
διπλότυπο ← Ψευδής
κ ← 0
Όσο όχι διπλότυπο και κ < Ν επανάλαβε
  κ ← κ + 1
  Αν Σ[Α[κ]] = 0 τότε
    Σ[Α[κ]] ← 1
  αλλιώς
    διπλότυπο ← Αληθής
  Τέλος_αν
Τέλος_επανάληψης
```

Η παραπάνω μέθοδος μπορεί να εφαρμοστεί μόνο όταν ο πίνακας παίρνει τιμές από ένα πεπερασμένο σύνολο (για να μπορούμε να δημιουργήσουμε έναν πεπερασμένο πίνακα Σ). Η τεχνική αυτή λέγεται “μέτρημα”, εμπνευσμένη από την *ταξινόμηση με μέτρημα*.

Άσκηση 89 - μεγαλύτερη συχνότητα

Ένα μονοδιάστατος πίνακας ακεραίων **Α[Ν]** περιέχει αποκλειστικά και μόνο φυσικούς αριθμούς από το 1 έως το 100. Γράψτε αλγόριθμο σε ψευδογλώσσα που να βρίσκει το στοιχείο (ή τα στοιχεία, αν είναι πολλά) του πίνακα με την μεγαλύτερη συχνότητα (αυτό που εμφανίζεται τις περισσότερες φορές).

Απάντηση

Θα κάνουμε μέτρημα των στοιχείων. Μετά σαρώνουμε τον πίνακα με τους μετρητές και βρίσκουμε το μέγιστο. Ξανά σάρωση του πίνακα των μετρητών για να βρούμε όλα τα “μέγιστα” και να τα εμφανίσουμε.

```
Αλγόριθμος ΜεγαλύτερηΣυχνότητα
Ν ← 5
Α[1] ← 2
Α[2] ← 5
Α[3] ← 2
```

```
A[4] ← 5
A[5] ← 1

! αρχικοποίηση του πίνακα συχνοτήτων
Για κ από 1 μέχρι Ν
  Σ[κ] ← 0
Τέλος_επανάληψης

! το μέτρημα γίνεται στον πίνακα Σ
Για κ από 1 μέχρι Ν
  Σ[Α[κ]] ← Σ[Α[κ]] + 1
Τέλος_επανάληψης

! η μέγιστη συχνότητα βρίσκεται από τον Σ
μεγ ← Σ[1]
Για κ από 2 μέχρι Ν
  Αν Σ[κ] > μεγ τότε μεγ ← Σ[κ]
Τέλος_επανάληψης

! τα αποτελέσματα προέρχονται από τον Σ
Για κ από 1 μέχρι Ν
  Αν Σ[κ] = μεγ τότε
    Εμφάνισε "Το στοιχείο ", κ, " εμφανίζεται ", μεγ, " φορές"
  Τέλος_αν
Τέλος_επανάληψης
Τέλος
```

Άσκηση 90 - διπλό-χ

Ένας πίνακας χαρακτήρων `Α[Ν]` περιέχει χαρακτήρες, αλλά έναν μόνο χαρακτήρα σε κάθε θέση. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ελέγχει αν υπάρχει μέσα στον πίνακα διπλό `χ`, δηλαδή τουλάχιστον δύο διπλανές θέσεις με τον χαρακτήρα `'χ'`. Δεν μας ενδιαφέρει αν υπάρχουν περισσότερα από 2 συνεχόμενα `χ`. Το αποτέλεσμα του ελέγχου να καταχωρείται σε μια λογική μεταβλητή `διπλόΧ`. Για παράδειγμα αν `Α=[β,χ,α,χ]` τότε `διπλοΧ=Ψευδής`, αλλά αν `Α=[χ,χ,α,χ]` τότε `διπλοΧ=Αληθής`. Ακόμη και για περισσότερα `χ` πρέπει πάλι να παίρνουμε την ίδια τιμή, για παράδειγμα αν `Α=[χ,χ,χ]` τότε `διπλοΧ=Αληθής`.

Απάντηση

Σαρώνουμε τον πίνακα μέχρι να βρούμε την πρώτη εμφάνιση διπλού `χ`.

```
Αλγόριθμος ΔιπλόΧΑλγ
Ν ← 4
Α[1] ← 'χ'
Α[2] ← 'χ'
Α[3] ← 'α'
Α[4] ← 'χ'

κ ← 0
διπλόΧ ← Ψευδής
Όσο όχι διπλόΧ και κ < Ν επανάλαβε
  κ ← κ + 1
  διπλόΧ ← Α[κ] = 'χ' και Α[κ+1] = 'χ'
Τέλος_επανάληψης

Εμφάνισε διπλόΧ
Τέλος
```

Ο παρατηρητικός αναγνώστης θα αναρωτιέται γιατί δεν κάνουμε κι αυτή την αναζήτηση σε μορφή προσπέρασης. Η απάντηση είναι ότι η διαδικασία προσπέρασης

δεν ταιριάζει παντού. Για να την χρησιμοποιήσουμε θα πρέπει να έχει νόημα η προσπέραση και να μπορούμε εύκολα να εξηγήσουμε (στον εαυτό μας και στους άλλους) τι ακριβώς προσπερνάμε. Εδώ για παράδειγμα, αν υποτεθεί ότι πάμε να προσπεράσουμε όλες τις θέσεις που δεν έχουν διπλό χ, θα καταλήξουμε στον κώδικα

```
κ ← 1
Όσο κ < Ν και (Α[κ] ≠ 'χ' ή Α[κ+1] ≠ 'χ') επανάλαβε    ! σύντομη αποτίμηση
  κ ← κ + 1
Τέλος_επανάληψης
διπλόΧ ← κ < Ν
```

ο οποίος είναι εντελώς δυσνόητος. Εφαρμόστε την διαδικασία προσπέρασης εκεί που πραγματικά ταιριάζει. Πάνω απ' όλα είναι η καθαρότητα του κώδικα όχι η επίδειξη τεχνικών.

**** Άσκηση 91 - αποκλειστικά διπλό-χ***

Η ίδια άσκηση με την προηγούμενη, αλλά τώρα ψάχνουμε αποκλειστικά και μόνο δύο χ στην σειρά και όχι τρία ή περισσότερα. Για παράδειγμα αν `Α=[χ,χ,χ]` τότε `διπλοΧ=Ψευδής`.

Απάντηση

Η καρδιά του αλγόριθμου είναι ίδια με την προηγούμενη άσκηση. Η διαφορά είναι ότι αφού βρούμε το πρώτο χ, με την *διαδικασία προσπέρασης* θα προσπερνάμε και θα μετράμε τα `χ` ταυτόχρονα. Αν είναι μόνο δύο τότε η αναζήτηση θεωρείται επιτυχής. (δείτε την άσκηση 73 για την διαδικασία προσπέρασης)

```
Αλγόριθμος ΑποκλειστικάΔιπλόΧ
Ν ← 4
Α[1] ← 'χ'
Α[2] ← 'α'
Α[3] ← 'χ'
Α[4] ← 'χ'

κ ← 1
διπλόΧ ← Ψευδής
Όσο κ < Ν και όχι διπλόΧ επανάλαβε
  Αν Α[κ] = 'χ' τότε                  ! αν βρεθεί χ
    κ ← κ + 1                         ! προχώρα μία θέση
    μετρ ← 1                          ! μέτρα ένα χ
    Όσο κ ≤ Ν και Α[κ] = 'χ' επανάλαβε ! προσπέραση και μέτρημα των χ
      μετρ ← μετρ + 1
      κ ← κ + 1
    Τέλος_επανάληψης
                      ! στο σημείο αυτό το μετρ δείχνει το πλήθος των χ
                      ! ενώ το κ δείχνει θέση η οποία σίγουρα δεν έχει χ
    διπλόΧ ← μετρ = 2 ! αν πετύχαμε το 2άρι, τότε όλα ΟΚ
  Τέλος_αν
  κ ← κ + 1
Τέλος_επανάληψης

Εμφάνισε διπλόΧ
Τέλος
```

Προσέξτε ότι τώρα ο αλγόριθμος έγινε γενικός. Μπορούμε να μετρήσουμε διπλά, τριπλά,... `χ`. Μπορούμε επίσης να περιλάβουμε και τις περιπτώσεις που δεν θέλουμε αποκλειστικότητα, απλά αλλάζοντας την συνθήκη `μετρ=2` με `μετρ≥2`.

Άσκηση 92 - μέτρημα διπλών χ με επικάλυψη

Ένας πίνακας χαρακτήρων `Α[Ν]` περιέχει χαρακτήρες, αλλά έναν μόνο χαρακτήρα σε κάθε θέση. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να μετράει πόσα διπλά `χ` υπάρχουν μέσα στον πίνακα. Η αλληλοεπικάλυψη επιτρέπεται, δηλαδή μια σειρά από τρία `χ` μετρούνται ως δύο εμφανίσεις `χχ`. Το αποτέλεσμα του μετρήματος να καταχωρείται σε μια ακέραια μεταβλητή `μέτρημαΧΧ`. Για παράδειγμα αν `Α=[φ,χ,χ,ψ]` τότε `μέτρημαΧΧ=1` ενώ αν `Α=[χ,φ,χ,χ,χ,ψ,χ,χ]` τότε `μέτρημαΧΧ=3`.

Απάντηση

```
Αλγόριθμος ΜέτρημαΔιπλόΧ
Ν ← 8
Α[1] ← 'χ'
Α[2] ← 'φ'
Α[3] ← 'χ'
Α[4] ← 'χ'
Α[5] ← 'χ'
Α[6] ← 'ψ'
Α[7] ← 'χ'
Α[8] ← 'χ'

μέτρημαΧΧ ← 0
Για κ από 1 μέχρι Ν - 1
  Αν Α[κ] = 'χ' και Α[κ + 1] = 'χ' τότε
    μέτρημαΧΧ ← μέτρημαΧΧ + 1
  Τέλος_αν
Τέλος_επανάληψης

Εμφάνισε μέτρημαΧΧ
Τέλος
```

Άσκηση 93 - μέτρημα διπλών χ χωρίς επικάλυψη

Η ίδια με την προηγούμενη άσκηση, αλλά να μην επιτρέπεται η αλληλοεπικάλυψη των `χ`. Αν κάποιο `χ` ανήκει σε ένα ζευγάρι και μετρήθηκε να μην μπορεί να μετρηθεί και σε άλλο ζευγάρι. Για παράδειγμα αν `Α=[χ,φ,χ,χ,χ,ψ,χ,χ]` τότε `μέτρημαΧΧ=2`.

Απάντηση

Ο κορμός της αναζήτησης παραμένει ο ίδιος. Το μόνο που αλλάζει είναι η μετακίνηση του δείκτη κάθε φορά που βρίσκουμε διπλό `χ`. Αντί να μετακινούμε τον δείκτη μία θέση δεξιά, τον μετακινούμε 2 θέσεις ώστε να προσπεράσουμε το `χ` που μετρήθηκε. Ακόμη ένα παράδειγμα που δείχνει ότι η αλλαγή του δείκτη σε μια επανάληψη `Για-από-μέχρι` πρέπει να επιτρέπεται (σε όλες τις σύγχρονες γλώσσες επιτρέπεται). Αν υποθέσουμε ότι επιτρέπεται, τότε στον προηγούμενο αλγόριθμο δεν έχουμε παρά να προσθέσουμε μια αύξηση του δείκτη κατά 1.

```
Για κ από 1 μέχρι Ν - 1
  Αν Α[κ] = 'χ' και Α[κ + 1] = 'χ' τότε
    κ ← κ + 1                          ! προσπέρνα το δεύτερο χ
    μέτρημαΧΧ ← μέτρημαΧΧ + 1
  Τέλος_αν
Τέλος_επανάληψης
```

Επειδή οι οδηγίες του υπουργείου δεν επιτρέπουν αλλαγή του δείκτη, υλοποιούμε την επανάληψη με `Όσο-επανάλαβε`.

```
μέτρημαΧΧ ← 0
κ ← 0
Όσο κ < Ν επανάλαβε
```

```
  κ ← κ + 1
  Αν Α[κ] = 'χ' και Α[κ + 1] = 'χ' τότε
    κ ← κ + 1
    μέτρημαΧΧ ← μέτρημαΧΧ + 1
  Τέλος_αν
Τέλος_επανάληψης

Εμφάνισε μέτρημαΧΧ
```

Άσκηση 94 - αναγραμματισμός

Δύο μονοδιάστατοι πίνακες **Α[Ν]** και **Β[Ν]**, περιέχουν δύο διαφορετικές φράσεις χαρακτήρα-χαρακτήρα (δηλαδή σε κάθε θέση του πίνακα είναι αποθηκευμένος ένας μόνο χαρακτήρας) με κεφαλαία γράμματα και χωρίς κενά. Να γραφεί αλγόριθμος σε ψευδογλώσσα που να ελέγχει αν η μία φράση είναι αναγραμματισμός της άλλης. Αναγραμματισμός μιας φράσης θεωρείται αυτή που χρησιμοποιεί ακριβώς τα ίδια γράμματα αλλά σε διαφορετικές θέσεις.

Απάντηση

Η σωστή λύση στο πρόβλημα του αναγραμματισμού είναι με μέτρημα των συχνοτήτων. Οι δύο πίνακες περιέχουν τιμές από ένα πεπερασμένο σύνολο (το σύνολο ASCII), συνεπώς μπορούμε να μετρήσουμε την συχνότητα εμφάνισης του κάθε χαρακτήρα, όπως στην άσκηση 89. Έπειτα συγκρίνουμε τους πίνακες συχνοτήτων μεταξύ τους. Αν είναι αναγραμματισμός, τότε πρέπει οι πίνακες συχνοτήτων να είναι πανομοιότυποι. Όλη η διαδικασία θα χρειαστεί μία σάρωση για τον κάθε πίνακα και άλλη μία για την σύγκριση των πινάκων συχνοτήτων.

Η ΓΛΩΣΣΑ και ψευδογλώσσα της ΑΕΠΠ δεν περιλαμβάνει εργαλεία για να μπορούμε να χειριστούμε τους χαρακτήρες ως πεπερασμένο σύνολο. Έτσι οι δυνατές λύσεις είναι δύο: (α) Με ταξινόμηση των δύο πινάκων. Μετά την ταξινόμηση πρέπει οι δύο πίνακες να ταυτίζονται, αλλιώς δεν είναι αναγραμματισμός. Θα πρέπει βέβαια να περιμένουμε να τελειώσουν δύο ταξινομήσεις, πριν αρχίσουμε πραγματικά να συγκρίνουμε. (β) Σημαντικά πιο αποδοτική είναι η εξαντλητική αναζήτηση. Ψάχνουμε έναν-έναν τους χαρακτήρες του **Α[]** στον **Β[]**. Κάθε φορά που βρίσκουμε έναν ίδιο χαρακτήρα, τον μαρκάρουμε (μέσα στον **Β**) με ένα ειδικό σύμβολο (*φρουρός* στην ορολογία του προγραμματισμού), ώστε να μην είναι δυνατόν να το ξανασυναντήσουμε. Σημαντικό κομμάτι του αλγορίθμου είναι η επιλογή του φρουρού. Θα πρέπει να είναι μια τιμή που να αποκλείεται να υπάρχει αρχικά στον πίνακα. Συνηθισμένη επιλογή για φρουρό-χαρακτήρα είναι το κενό αλφαριθμητικό (άνοιγμα και αμέσως κλείσιμο εισαγωγικών, χωρίς να γράψουμε τίποτα ανάμεσα στα εισαγωγικά, **“”** ή **''**). Η πρώτη φορά που δεν θα βρούμε έναν χαρακτήρα μέσα στον **Β**, θα σηματοδοτήσει την αποτυχία του ελέγχου. Αντίθετα αν όλα πάνε καλά μέχρι τέλους, τότε σημαίνει ότι οι φράσεις είναι αναγραμματισμός.

```
Αλγόριθμος Αναγραμματισμός
Ν ← 7
Α[1] ← 'π'
Α[2] ← 'α'
Α[3] ← 'ρ'
Α[4] ← 'α'
Α[5] ← 'λ'
Α[6] ← 'ι'
Α[7] ← 'α'
```

```
B[1] ← 'λ'
B[2] ← 'π'
B[3] ← 'ρ'
B[4] ← 'ι'
B[5] ← 'α'
B[6] ← 'α'
B[7] ← 'α'

αναγραμ ← Αληθής
κ ← 1                                  ! το κ σαρώνει τον Α
Όσο αναγραμ και κ ≤ Ν επανάλαβε
  λ ← 1                                ! το λ σαρώνει τον Β
  Όσο λ ≤ Ν και Α[κ] ≠ Β[λ] επανάλαβε ! προσπέρνα όσα δεν ταιριάζουν
    λ ← λ + 1
  Τέλος_επανάληψης
  Αν λ < Ν + 1 τότε                    ! αν δεν βγήκαμε έξω από τον Β ..
    Β[λ] ← ''                          ! ..τότε βρέθηκε και το μαρκάρουμε
  αλλιώς
    αναγραμ ← Ψευδής                   ! αλλιώς, τέλος σε όλα
  Τέλος_αν
  κ ← κ + 1
Τέλος_επανάληψης
Εμφάνισε αναγραμ
Τέλος
```