Παρατηρήσεις πάνω στο lab4

Επιλογή κατάλληλων δομών:

Οι δομές επανάληψης της C μπορούν γενικά να χρησιμοποιηθούν η μία στη θέση της άλλης, αλλά κατά κανόνα χρησιμοποιούμε for όταν γνωρίζουμε τον αριθμό επαναλήψεων, και while ή do-while όταν δεν τον γνωρίζουμε. Επιλέγουμε do-while έναντι while όταν ο κώδικας προς επανάληψη πρέπει να εκτελεστεί τουλάχιστον μία φορά. Επομένως, στην πρώτη άσκηση, στο σημείο που δίνεται το Reserved time, η πιο κατάλληλη δομή επανάληψης είναι η do-while ενώ στη δεύτερη άσκηση η for.

Χρήση continue/break:

Η χρήση continue ή/και break είναι συχνά απαραίτητη για τη ροή του προγράμματος, αλλά όχι πάντα. Να τις χρησιμοποιείτε όταν χρειάζεται, αλλά όχι όταν είναι περιττές. Η δικαιολογία "το έβαλα για να φαίνεται καλύτερα" δεν είναι καλή. Για παράδειγμα, στον παρακάτω κώδικα ΔΕΝ θα έπρεπε να υπάρχει continue:

```
while (cond) {
    if (cond1) {
        statement1;
        continue;    // ΔΕΝ ΘΑ ΕΠΡΕΠΕ ΝΑ ΥΠΑΡΧΕΙ
    }
    else {
        statement2;
        continue;    // ΔΕΝ ΘΑ ΕΠΡΕΠΕ ΝΑ ΥΠΑΡΧΕΙ
    }
}
```

if/else:

Όταν χρησιμοποιείτε if/else μην κάνετε περιττούς ελέγχους. Είδαμε συχνά το παρακάτω:

```
if (j%2 == 0) {
    ...
}
else if (j%2 == 1) { // ΆΣΚΟΠΟΣ ΕΛΕΓΧΟΣ!
     ...
}
```

Το j είναι ή μονό ή ζυγό. Αν δεν είναι ζυγό, τότε αναγκαστικά είναι μονό, επομένως αρκεί το:

```
if (j%2 == 0) {
    ...
}
else {
    ...
}
```

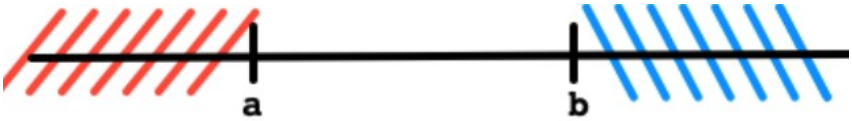
Έλεγχοι εγκυρότητας:

Είδαμε αρκετά λάθη στους ελέγχους καθώς και αρκετές περιπτώσεις φοιτητών που αντί να σκεφτούν τι είναι σωστό, δοκίμαζαν διάφορους τελεστές (|| ή &&, < ή <=) μέχρι να φανεί ότι δουλεύει το πρόγραμμα. Αυτό είναι λάθος τακτική.

Αν δεν είστε σίγουροι πώς να σχηματίσετε μια σύνθετη συνθήκη, δοκιμάστε να ζωγραφίσετε το εμπλεκόμενο διάστημα στο χαρτί.

Ας υποθέσουμε ότι έχουμε δύο αριθμούς a και b με a < b.

Ας εξετάσουμε τις σύνθετες συνθήκες που περιλαμβάνουν τις εκφράσεις x < a, x > b για κάποιο x. Σκιαγραφούμε με κόκκινο την περιοχή που είναι αληθής η έκφραση x < a και με μπλε την περιοχή που είναι αληθής η έκφραση x > b.

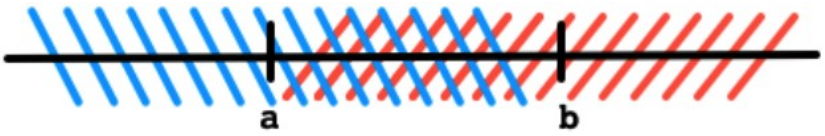


Ο τελεστής || έχει αποτέλεσμα "αληθές" αν οποιαδήποτε από τις εκφράσεις που συνδέει είναι αληθής. Με άλλα λόγια, η συνθήκη `x < a || x > b` είναι αληθής σε όλα τα σημεία που ανήκουν σε σκιαγραφημένη περιοχή, ανεξαρτήτως αν αυτή είναι μπλε ή κόκκινη.

Ο τελεστής && έχει αποτέλεσμα "αληθές" αν και οι δύο εκφράσεις που συνδέει είναι αληθείς. Με άλλα λόγια, η συνθήκη `x < a && x > b` είναι αληθής σε όλα τα σημεία που ανήκουν σε κοινή σκιαγραφημένη περιοχή. Σε περιοχή που είναι ταυτόχρονα μπλε και κόκκινη. Όπως φαίνεται από το σχήμα, η έκφραση `x < a && x > b` δε θα είναι ποτέ αληθής.

Ένας απλός τρόπος να αποφύγετε λάθη όπως η έκφραση `x < a && x > b` είναι να προσπαθήσετε να σκεφτείτε έναν αριθμό για τον οποίο η έκφραση είναι αληθής. Θα δείτε σύντομα πως αυτό δε γίνεται.

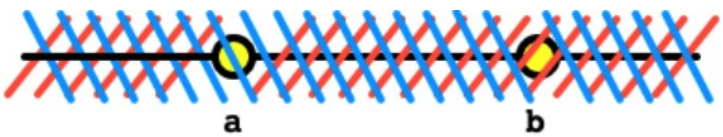
Ας υποθέσουμε τώρα ότι για τους ίδιους αριθμούς μας ενδιαφέρει να εξετάσουμε τις σύνθετες συνθήκες που περιλαμβάνουν τις εκφράσεις x > a, x < b για κάποιο x. Σκιαγραφούμε με κόκκινο την περιοχή που είναι αληθής η έκφραση x > a και με μπλε την περιοχή που είναι αληθής η έκφραση x < b.



Όπως είπαμε πιο πάνω, ο τελεστής || έχει αποτέλεσμα "αληθές" αν οποιαδήποτε από τις εκφράσεις που συνδέει είναι αληθής. Με άλλα λόγια, η συνθήκη `x > a || x < b` είναι αληθής σε όλα τα σημεία που ανήκουν σε σκιαγραφημένη περιοχή, ανεξαρτήτως αν αυτή είναι μπλε ή κόκκινη. Άρα, αυτή η έκφραση είναι αληθής για κάθε τιμή του x!

Ο τελεστής && έχει αποτέλεσμα "αληθές" αν και οι δύο εκφράσεις που συνδέει είναι αληθείς. Με άλλα λόγια, η συνθήκη x > a && x < b είναι αληθής σε όλα τα σημεία που ανήκουν σε κοινή σκιαγραφημένη περιοχή. Όπως φαίνεται από το σχήμα, τα κοινά σημεία είναι αυτά που βρίσκονται ανάμεσα στο a και στο b.

Τέλος, ας εξετάσουμε τις σύνθετες συνθήκες που περιλαμβάνουν τις εκφράσεις x != a, x != b για κάποιο x. Σκιαγραφούμε με κόκκινο την περιοχή που είναι αληθής η έκφραση x != a και με μπλε την περιοχή που είναι αληθής η έκφραση x != b. Επιπλέον, με κίτρινο χρώμα έχουμε τονίσει τα σημεία του άξονα όπου βρίσκονται τα σημεία a, b.



Ο τελεστής || έχει αποτέλεσμα "αληθές" αν οποιαδήποτε από τις εκφράσεις που συνδέει είναι αληθής. Με άλλα λόγια, η συνθήκη `x != a || x != b` είναι αληθής σε όλα τα σημεία που ανήκουν σε σκιαγραφημένη περιοχή, ανεξαρτήτως αν αυτή είναι μπλε ή κόκκινη. Άρα, αυτή η έκφραση είναι αληθής για κάθε τιμή του x! (Αν το καλοσκεφτούμε, είναι λογικό. Η συνθήκη δε μπορεί να είναι ποτέ ψευδής γιατί το x δε μπορεί ποτέ να είναι ταυτόχρονα ίσο με το a και το b).

Ο τελεστής && έχει αποτέλεσμα "αληθές" αν και οι δύο εκφράσεις που συνδέει είναι αληθείς. Με άλλα λόγια, η συνθήκη `x !=a && x != b` είναι αληθής σε όλα τα σημεία που ανήκουν σε κοινή σκιαγραφημένη περιοχή. Σε περιοχή που είναι ταυτόχρονα μπλε και κόκκινη. Όπως φαίνεται από το σχήμα, η έκφραση είναι αληθής σε όλα τα σημεία εκτός από τα κίτρινα. Σημειώστε πως στα ελληνικά λέμε πως "το x δεν είναι ούτε a ούτε b".

Κάτι άλλο που έχει ενδιαφέρον στην πρώτη άσκηση είναι η σειρά των ελέγχων. Θέλουμε να εξετάσουμε τις παρακάτω περιπτώσεις:

Συνθήκη τερματισμού: Απλός έλεγχος για μηδέν. Είναι πρακτικό να ελεγθεί πρώτα, ώστε να τερματίσει η επανάληψη εάν χρειάζεται.

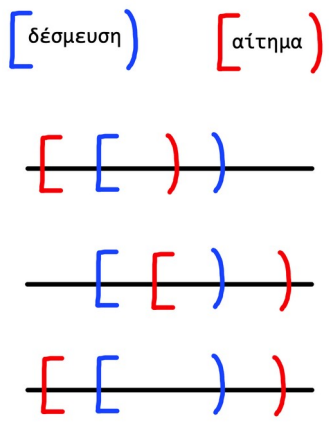
Άκυρη είσοδος: Επίσης εύκολες συνθήκες και είναι πιο πρακτικό ο έλεγχος να γίνει μετά τον έλεγχο τερματισμού ώστε να συνεχίσουμε στην επόμενη επανάληψη εάν χρειάζεται.

Επικαλύψεις: Εδώ υπάρχουν αρκετές περιπτώσεις και χρειάζεται περισσότερη σκέψη. Κατ'αρχήν, το πιο απλό είναι να ελέγξουμε εάν δεν υπάρχει επικάλυψη. Αυτό ισχύει αν η ώρα έναρξης του αιτήματος προκύπτει μετά ή ακριβώς κατά τη λήξη του διαστήματος δέσμευσης, ή η ώρα λήξης του αιτήματος προκύπτει πριν ή ακριβώς κατά την ώρα έναρξης. Προσοχή στα άκρα: Το διάστημα δέσμευσης είναι κλειστό στην έναρξη και ανοιχτό στη λήξη. Αυτό επηρεάζει το πότε χρησιμοποιούμε <= , >= έναντι < , > .

Σε κάθε άλλη περίπτωση υπάρχει επικάλυψη και μας ενδιαφέρει να δούμε αν αυτή είναι πλήρης ή μερική. Και πάλι επιλέγουμε να εξετάσουμε την πιο εύκολη περίπτωση που είναι η πλήρης επικάλυψη. Θα πρέπει να δούμε αν η ώρα έναρξης του αιτήματος προκύπτει μετά ή ακριβώς κατά την έναρξη του διαστήματος δέσμευσης και η ώρα λήξης πριν ή ακριβώς κατά τη λήξη του διαστήματος δέσμευσης. Και πάλι δίνουμε προσοχή στο αν και πότε χρησιμοποιούμε = στις ανισότητες.

Εάν δεν ισχύει τίποτα από τα παραπάνω, τότε προφανώς έχουμε μερική επικάλυψη, άρα απλά εκτυπώνουμε maybe μέσα σε μια else χωρίς να χρειάζεται να κάνουμε κάποιο έλεγχο.

Ο παραπάνω τρόπος σκέψης δίνει την πιο απλή λύση. Εάν παρόλα αυτά επιλέξουμε να κάνουμε ειδικό έλεγχο για την περίπτωση του maybe, τότε το πρώτο βήμα είναι να σχεδιάσουμε τις περιπτώσεις στο χαρτί για να δούμε πώς μπορούμε να γράψουμε μία απλή συνθήκη που να τις καλύπτει όλες:



Εμφανίζουμε με μπλε χρώμα το διάστημα που είναι ήδη δεσμευμένη η αίθουσα και με κόκκινο το διάστημα του αιτήματος.

Εξετάζοντας τις σχέσεις που έχουν μεταξύ τους άκρα των δύο διαστημάτων, παρατηρούμε πως σε όλες τις περιπτώσεις *η έναρξη του αιτήματος είναι πριν τη λήξη της δέσμευσης* **και** η *λήξη του αιτήματος είναι μετά την έναρξη της δέσμευσης*.

Επιπλέον, η παραπάνω συνθήκη δεν ισχύει στην περίπτωση που δεν υπάρχει επικάλυψη.

Άρα, αυτή η συνθήκη είναι ικανή και αναγκαία για την ύπαρξη επικάλυψης. Το μόνο "πρόβλημα" είναι πως ισχύει και στην περίπτωση πλήρους επικάλυψης, αλλά αυτό λύνεται κάνοντας πρώτα τον έλεγχο για πλήρη επικάλυψη και φροντίζοντας ώστε ο έλεγχος της μερικής επικάλυψης να γίνεται μόνο αν ο έλεγχος της πλήρους αποτύχει.

Εναλλαγή χαρακτήρων στη δεύτερη άσκηση

Υπάρχουν αρκετοί τρόποι για να υλοποιηθεί η δεύτερη άσκηση. Αρκετοί από εσάς σκεφτήκατε έξυπνες λύσεις. Αναφέρουμε εδώ δύο από τους πιο συνοπτικούς τρόπους επίλυσης του προβλήματος.

Σε κάθε περίπτωση, θέλουμε να εκτυπώσουμε τους χαρακτήρες ανά γραμμή:

```
for (i=0; i<SIZE; i++) {
    for (j=0; j<SIZE; j++) {
        // έλεγχος κι εκτύπωση
    }
}
```

Ένας τρόπος να υλοποιήσουμε την εναλλαγή των χαρακτήρων είναι παρατηρώντας ότι στις ζυγές γραμμές ξεκινάμε από τον ένα και στις μονές από τον άλλο. Επομένως αρκεί ένας έλεγχος για το αν το i%2 είναι μηδέν, κι αναλόγως η αντίστοιχη printf η οποία εκτυπώνει 2 χαρακτήρες τη φορά.

Ένας δεύτερος τρόπος είναι μετά την εκτύπωση μιας γραμμής, να εναλλάσσουμε (swap) τους 2 χαρακτήρες με τη βοήθεια μια τρίτης μεταβλητής ή με χρήση του τελεστή ^ (xor).

Στοίχιση, ονόματα μεταβλητών, κενές γραμμές:

RTFM.

Επιπλέον, να αποφεύγετε τεράστιες γραμμές γιατί δε διαβάζονται. Το παρακάτω είναι ένα παράδειγμα προς αποφυγή:

```
52              if ((arxiki_ora < arxiki_ora_desmevmeni && teliki_ora < teliki_ora_desmevmeni && teliki_ora >
_   arxiki_ora_desmevmeni) || (arxiki_ora > arxiki_ora_desmevmeni && teliki_ora < teliki_ora_desmevmeni) || (arxiki_ora
_   < arxiki_ora_desmevmeni && teliki_ora > teliki_ora_desmevmeni) || (arxiki_ora == teliki_ora_desmevmeni &&
_   teliki_ora > teliki_ora_desmevmeni) || (arxiki_ora < arxiki_ora_desmevmeni && teliki_ora == arxiki_ora_desmevmeni))
_   {
53                  printf("%d-%d maybe.\n", arxiki_ora, teliki_ora);
```

Όπως είδατε στην προηγούμενη ενότητα, η συνθήκη θα μπορούσε να είναι πιο απλή, αλλά ακόμη και μια τέτοια σύνθετη συνθήκη θα μπορούσε να είχε γραφτεί πιο καθαρά ως εξής:

```
52              if ((arxiki_ora < arxiki_ora_desmevmeni
53                  && teliki_ora < teliki_ora_desmevmeni
54                  && teliki_ora > arxiki_ora_desmevmeni)
55                  || (arxiki_ora > arxiki_ora_desmevmeni && teliki_ora < teliki_ora_desmevmeni)
56                  || (arxiki_ora < arxiki_ora_desmevmeni && teliki_ora > teliki_ora_desmevmeni)
57                  || (arxiki_ora == teliki_ora_desmevmeni && teliki_ora > teliki_ora_desmevmeni)
58                  || (arxiki_ora < arxiki_ora_desmevmeni && teliki_ora == arxiki_ora_desmevmeni)) {
59                  printf("%d-%d maybe.\n", arxiki_ora, teliki_ora);
```

Παρατηρήστε πως οι τελεστές || και && εμφανίζονται στην αρχή κάθε γραμμής ώστε να είναι φανερό πως πρόκειται για συνέχεια από την προηγούμενη.