Pascal: Στοίχιση

Η στοίχιση δεν επηρεάζει με κανένα τρόπο το αν είναι συντακτικά σωστό το πρόγραμμα (δηλαδή αν θα βρει κάποιο λάθος ο compiler).

Η στοίχιση βοηθάει στο να γίνει πιο ευανάγνωστο το πρόγραμμα και στην εύρεση των λαθών στα προγράμματα. Πράγματι, το 70% των σφαλμάτων προέρχονται από κάποια `end` τα οποία λείπουν ή περισσεύουν ή από ερωτηματικά που δεν μπαίνουν. Τα λάθη αυτού του τύπου είναι γίνονται εμφανή άμεσα αν τηρηθούν κάποιοι κανόνες στοίχισης.
Επιπλέον, ένα θέμα στις εξετάσεις είναι πάντα η επανεγγραφή ενός προγράμματος με στοίχιση και σχόλια οπότε καλόν είναι να είμαστε έτοιμοι.

Το παρακάτω «καλλιτεχνικό» πρόγραμμα...

```
Program ArrayDemo(input,output); Var Counter, Sum : integer; Pinakas : array[1..10] of
integer; begin Counter:=0;Writeln ('Dwse 10 thetikous arithmous');Writeln; repeat
Counter:=Counter+1; repeat {Emfanizei dwse ton 1o,2o... thetiko arithmo} Write ('Dwse
ton ',Counter,'o thetiko arithmo : '); readln(Pinakas[Counter]); until(Pinakas[Counter
]>0); until

(Counter=10);          Counter:=0;

               Sum:=0;
repeat                          Counter:=
   Counter + 1;       Sum:=Sum +
           Pinakas[Counter];

until (Counter=10); writeln; Writeln('To a8roisma twn 10 ari8mwn einai : ', Sum );end.
```

είναι ακριβώς ίδιο και εξίσου συντακτικά ορθό με αυτό που ακολουθεί:

```
Program ArrayDemo(input,output);
Var Counter, Sum : integer;
    Pinakas : array[1..10] of integer;
  begin
     Counter:=0;
     Writeln ('Dwse 10 thetikous arithmous');Writeln;
     repeat
        Counter:=Counter + 1;
        repeat
           {Emfanizei dwse ton 1o,2o... thetiko arithmo}
           Write ('Dwse ton ',Counter,'o thetiko arithmo : ');
           readln(Pinakas[Counter]);
        until (Pinakas[Counter]>0);
     until (Counter=10);
     Counter:=0; Sum:=0;
     repeat
        Counter:=Counter + 1;
        Sum:=Sum + Pinakas[Counter];
     until (Counter=10);

     writeln; Writeln('To a8roisma twn 10 ari8mwn einai : ', Sum );
  end.
```

Παρόλα αυτά δεν νομίζω ότι θα ήθελε κανείς να διαβάσει το πρώτο, όπως επίσης είναι πάρα πολύ δύσκολο να βρεθούν τα λάθη. Για παράδειγμα, αν ο compiler έδινε λάθος στην $3^{η}$ γραμμή, αυτό δεν θα έλεγε τίποτα στην πρώτη περίπτωση, ενώ στη δεύτερη θα καταδείκνυε σαφώς το λάθος. Επίσης η θέση των `begin` και `end` όπως και τα αντίστοιχα blocks κώδικα που ορίζουν τα `repeat … until`, είναι τόσο προφανή στη δεύτερη περίπτωση που δεν αφήνουν περιθώρια λάθους.

Θα πρέπει προκαταβολικά να πούμε ότι δεν έχουν οριστεί παγκόσμιοι κανόνες σχετικά με τη στοίχιση. Αυτό που σε κάποιον μπορεί να φαίνεται σωστό μπορεί σε κάποιον άλλο να φαίνεται απαράδεκτο.

Ας δούμε λοιπόν σε ποιοι είναι οι βασικοί κανόνες στοίχισης όπως αυτοί ζητούνται στα εργαστήρια.

1. Όταν θέλουμε να κάνουμε μία εσοχή στον κώδικα δίνουμε 3 επιπλέον κενά στην αρχή της γραμμής.
2. Εσοχή γίνεται πάντα για μία εντολή (ή ένα block που ορίζεται από `begin`…`end` ή `repeat`…`until`) μετά τις εντολές `then`, `else`, `while`, `do`, `of`.
3. Εσοχή γίνεται στον κώδικα ανάμεσα στις εντολές `begin … end`, `repeat ... until`

Οι παραπάνω 3 κανόνες καλύπτουν τις περισσότερες απλές περιπτώσεις και δίνουν ευανάγνωστα προγράμματα με λιγότερα λάθη.

Pascal: Φόρμες

Υπάρχει σε κάθε γλώσσα ένα σύνολο συμβόλων που εμφανίζονται πάντα σε συγκεκριμένη μορφή. Όποτε ξεχνάτε ένα στοιχείο κάποιας φόρμας παράγονται αλυσιδωτά λάθη που μπορεί να είναι και αρκετές σελίδες. Οι κύριες φόρμες στην pascal είναι οι εξής:

```
begin … end;
“ … “
‘ … ‘
( … )
{ … }
[ … ]
if ( … ) then … else …
for ( … ) to … do …
while (…) do …
repeat … until (…)
```

Αυτά τα ζεύγη γράφονται ΠΑΝΤΑ προκαταβολικά ως φόρμα και μετά συμπληρώνουμε τα ενδιάμεσα στοιχεία. Δηλαδή γράφουμε:

```
if () then else
```

και μετά ερχόμαστε και συμπληρώνουμε τα κενά...

```
if (a>0) then a:= -a else a:=a;
```

ή στο πιο σύνηθες παράδειγμα γράφουμε:

```
begin
end;
```

και μετά συμπληρώνουμε...

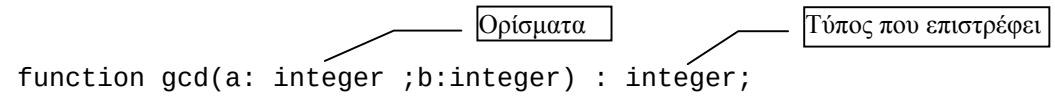
```
begin
   writeln(a);
   writeln(b);
end;
```

Με αυτόν τον τρόπο ξεχνάμε μια για πάντα τα γνωστά λάθη ρουτίνας που προκαλούν οι ελλιπείς φόρμες.

Pascal: Ευκλείδιος Αλγόριθμος Εύρεσης του ΜΚΔ δύο Ακεραίων

Η λύση στο πρόβλημα της εύρεσης του Μέγιστου Κοινού Διαιρέτη είναι η παρακάτω function.

```
function gcd(a: integer ;b:integer) : integer;
var i,j:integer;
begin
i:=abs(a);
j:=abs(b);
while (i>0) and (j>0) do
if i>=j then i:=i mod j else j:=j mod i;
gcd:=i+j;
end;
```



Η δήλωση της function GCD: Παίρνει ως ορίσματα δύο ακεραίους α και β και επιστρέφει την ακέραια τιμή του μέγιστου κοινού διαιρέτη. GCD : $IN^2 \rightarrow IN$

```
var i,j:integer;
```

Οι δύο μεταβλητές i και j δηλώνονται μέσα στη function και χαρακτηρίζονται τοπικές. Οι τοπικές μεταβλητές υπάρχουν μόνο μέσα στη συγκεκριμένη function και δεν απασχολούν το υπόλοιπο πρόγραμμα.

```
i:=abs(a); j:=abs(b);
```

Οι i και j πρέπει να είναι θετικές για να λειτουργεί ο παρακάτω αλγόριθμος. Η μετατροπή γίνεται με την abs(). (απόλυτη τιμή).

```
while (i>0) and (j>0) do
if i>=j then i:=i mod j else j:=j mod i;
```

Ο ευκλείδιος αλγόριθμος... Απλά δουλεύει. Για περισσότερα ρωτήστε τον Ευκλείδη.

```
gcd:=i+j;
```

Έτσι ορίζεται η τιμή που θα επιστρέψει η function (i + j). Από τον ευκλείδειο αλγόριθμο, το άθροισμα αυτό είναι ο μέγιστος κοινός διαιρέτης.

```
var a,b:integer;
begin
readln(a,b);
writeln(O megistos koinos
                diairetis twn ',a,' kai ',b , ' einai ' ,gcd(a,b));
end.
```

Παράδειγμα προγράμματος που χρησιμοποιεί την παραπάνω function.

Με τη μέθοδο αυτή μπορεί να γίνει εύκολα απλοποίηση κλασμάτων:

```
gc:=gcd(Ar,Par);
Ar:=Ar div gc;
Par:=Par div gc;
```

* div είναι η ακέραια διαίρεση

Pascal: Η εντολή for

Μια πάρα πολύ χρήσιμη εντολή που δημιουργεί δυστυχώς αρκετή σύγχυση στα εργαστήρια είναι η εντολή `for`. Η εντολή αυτή είναι μια συντόμευση της αντίστοιχης διαδικασίας με `while` που υιοθετήθηκε λόγο της πάρα πολύ συχνής χρήσης της.

Συγκεκριμένα:

```
for i:=1 to 10 do writeln(i);
```

ισοδυναμεί με

```
i:=1;
while (i <= 10) do
   begin
      writeln(i);

      i:= i +1;
   end;
```

και γενικότερα:

```
for i:=a to b do
   begin
      …
   end;
```

ισοδυναμεί με

```
i:=a;
while (i <= b) do
   begin
      begin
         …
      end;
      i:= i +1;
   End;
```

Έχοντας την παραπάνω ισοδυναμία στο μυαλό, (ή στο χαρτί την ώρα των εξετάσεων) είσαι σε θέση να απαντήσεις με ευκολία σε ερωτήματα όπως το αν και πόσες επαναλήψεις στο `begin … end` θα γίνουν κ.ο.κ.

Συνοπτικά η παραπάνω εντολή κάνει μέτρημα με τη μεταβλητή i στα προκαθορισμένα όρια που της δίνουμε. Αν θέλουμε αντίστροφη μέτρηση υπάρχει και η παρακάτω εκδοχή της η οποία βέβαια χρησιμοποιείται σπάνια.

```
for i:=a downto b do
   begin
      …
   end;
```

ισοδυναμεί με:

```
i:=a;
while (i >= b) do
   begin
      begin
         …
      end;
      i:= i -1;
   end;
```

Pascal: Πως βρίσκουμε τα λάθη

Η μεθοδολογία για την εύρεση λαθών (debugging) είναι γενική για όλες της γλώσσες προγραμματισμού. Αυτό που διαφοροποιείται είναι τα επιπλέον βοηθητικά εργαλεία που μας δίνει η κάθε γλώσσα για τον εντοπισμό τους. Η pascal που έχουμε στο εργαστήριο δεν δίνει κανένα βοηθητικό εργαλείο για τον εντοπισμό λαθών εκτός από την (συνήθως τεράστια) λίστα με σφάλματα κατά το compile. Αυτό μας οδηγεί στον εντοπισμό λαθών με τις πλέον γενικές μεθοδολογίες που περιγράφουμε παρακάτω.

Καταρχάς υπάρχουν δύο κατηγορίες σφαλμάτων. Τα **συντακτικά σφάλματα** και τα **λογικά σφάλματα**.

Τα συντακτικά σφάλματα είναι τα εύκολα και είναι αυτά που επισημαίνει ο compiler π.χ. Στο πρόγραμμα `begin writern('a') end` θα μας πει ότι δεν βρίσκει την εντολή `writern`.

Τα λογικά σφάλματα είναι τα πιο δύσκολα και είναι αυτά τα οποία φανερώνονται μετά το compile και κατά την εκτέλεση του προγράμματος. Κι εδώ υπάρχουν δύο υποκατηγορίες, τα σφάλματα τυπικής εκτέλεσης και τα σφάλματα οριακών συνθηκών.

Η πρώτη κατηγορία είναι αυτή που τα λάθη γίνονται προφανή σε κάθε εκτέλεση του προγράμματος π.χ. ενώ εμείς θέλουμε το πρόγραμμά μας να εμφανίζει θετικό ακέραιο, αυτό επιστρέφει αρνητικό μιγαδικό (!). Τα σφάλματα αυτά είναι πιο δύσκολα από τα προηγούμενα.

Τα σφάλματα οριακών συνθηκών είναι αυτά τα οποία συμβαίνουν μόνο για ορισμένες ειδικές τιμές των παραμέτρων του προγράμματος. Αυτά τα λάθη είναι τα πιο δύσκολα απ' όλα, πολλές φορές είναι αδύνατο να εντοπιστούν και γενικά δεν μας απασχολούν σε αυτή τη φάση. Παράδειγμα τέτοιου λάθους ήταν το λάθος σε μία γραμμή προγράμματος Fortran που οδήγησε διαστημόπλοιο της NASA στην καταστροφή. Άλλο πιο απτό παράδειγμα είναι όταν γράφουμε χαρακτήρες αντί για νούμερα ως τιμή σε μία εντολή `readln(a)` όπου a `integer` ή η απρόβλεπτη συμπεριφορά όταν δοθεί αρνητικός αριθμός εκεί που περιμένουμε θετικό.

Οι γενικές μεθοδολογίες είναι οι εξής:

1. Συντακτικά σφάλματα

α) Στοίχιση

Το θέμα αυτό καλύφθηκε παραπάνω.

β) Συχνό compile

Η κύρια πηγή των συντακτικών λαθών είναι ότι γράφουμε πολλές γραμμές κώδικα με τη μια από μνήμης και μετά τον ελέγχουμε. Η λύση είναι να γράφουμε το πρόγραμμα λίγο λίγο (π.χ. κάθε 5 γραμμες να κάνουμε compile). Αυτή είναι μία συνήθεια που αποκτιέται με τον καιρό.

γ) Βαθμιαία αποκάλυψη

Αυτή η τεχνική είναι η πιο ισχυρή και βρίσκει όλα τα συντακτικά λάθη. Επειδή είναι κάπως επίπονη, πρέπει να χρησιμοποιείται μόνο όταν τα σφάλματα δεν είναι προφανή. Υπάρχουν περιπτώσεις που βρίσκει ο compiler λάθη π.χ. στην γραμμή 39 ενώ μετά την προσεκτική εξέταση του προγράμματος είναι 100% σίγουρο ότι δεν υπάρχουν σφάλματα εκεί. Αυτό συμβαίνει γιατί στην γραμμή 39 εκδηλώνεται σφάλμα το οποίο έχουμε κάνει πιο πάνω. Είναι σαν το λάθος στην πρόταση:

«Το ΣΕΜΦΕ είναι σχολή του είναι σχολή του Πολυτεχνείου.»

Το λάθος που θα έβγαζε ένας συντακτικός αναλυτής είναι ότι το «του» δεν μπορεί να είναι υποκείμενο: [... του είναι σχολή του Πολυτεχνείου]. Παρ' όλα αυτά το πραγματικό σφάλμα βρίσκεται πιο πριν.

Για αυτού του τύπου τα λάθη η παρακάτω μέθοδος είναι ιδανική.

Βάζουμε σε σχόλια ({...}) όλο το πρόγραμμα που γράψαμε μετά το τελευταίο επιτυχημένο compile. Με αυτό τον τρόπο το μέρος του προγράμματος που εμφανίζει πρόβλημα γίνεται ανενεργό. Αποκαλύπτουμε μία μία τις εντολές (ή τα blocks που ορίζονται από `begin… end` ή `repeat … until`) μετακινώντας τις αγκύλες, δηλαδή βγάζοντας τα σχόλια από αυτές. Κάνουμε compile και βλέπουμε αν εμφανίζονται λάθη. Αν πράγματι υπάρχουν τα διορθώνουμε εύκολα αφού ξέρουμε ακριβώς που βρίσκονται (στην εντολή που μόλις αποκαλύψαμε). Επαναλαμβάνουμε την παραπάνω διαδικασία μέχρι να αποκαλυφθεί όλος ο κώδικας.

Ένα πολύ απλό παράδειγμα που προκύπτει από λάθος στοίχιση είναι το εξής:

```
1     Program cba(input,output);
2     begin
3        begin;
4           write('c');
5           begin;
6              write('b');
7              repeat
8                 write('a');
9              until(true);
10             end;
11          end;
12       end;                < Error 94: "." expected.
13    end.
```

Αυτό εμφανίζει λάθος στη σειρά 12 όπου προφανώς δεν υπάρχει κανένα συντακτικό λάθος. Εφαρμόζουμε λοιπόν την παραπάνω μεθοδολογία

```
1     Program cba(input,output);
2     begin
3        {begin;
4           write('c');
5           begin;
6              write('b');
7              repeat
8                 write('a');
9              until(true);
10             end;
11          end;
12       end;}
13    end.
```

Βήμα 1ο Το πρόγραμμα δεν εμφανίζει σφάλμα

```
1     Program cba(input,output);
2     begin
3        begin;
4           write('c');
5           {begin;
6              write('b');
7              repeat
8                 write('a');
9              until(true);
10             end;
11          end; }
12       end;
13    end.
```

Βήμα 2ο Το πρόγραμμα δεν εμφανίζει σφάλμα

```
1     Program cba(input,output);
2     begin
3        begin;
4           write('c');
5           begin;
6              write('b');
7              {repeat
8                 write('a');
9              until(true);
10             end;}
11          end;
12       end;
13    end.
```

Βήμα 3ο Το πρόγραμμα δεν εμφανίζει σφάλμα

```
1     Program cba(input,output);
2     begin
3        begin;
4           write('c');
5           begin;
6              write('b');
7              repeat
8                 {write('a'); }
9              until(true);
10             end;
11          end;
12       end;
13    end.
```

Βήμα 4ο Το πρόβλημα εμφανίζεται.

Μετά λοιπόν την παραπάνω ανάλυση προφανώς το πρόβλημα εμφανίζεται μεταξύ των βημάτων 3 και 4 δηλαδή στον κώδικα

```
7               repeat
8                  ...
9               until(true);
10              end;
```

Νομίζω ότι είναι πλέον προφανές ότι το πρόβλημα οφείλεται στο παραπάνω `end` μετά το `until`. Αφαιρώντας το δεν εμφανίζονται περαιτέρω σφάλματα κατά το compile.

δ) Μια πολύ ωραία και γρήγορη μέθοδος που εμμέσως πρότεινε κάποιος φοιτητής στα εργαστήρια για ανίχνευση ειδικά λαθών στα `begin … end` ή `repeat … until` είναι η εξής. Μετρήστε όλα τα `begin` του προγράμματος σας και όλα τα `end`. Αν σας βγει διαφορετικό νούμερο, τότε έχετε σίγουρα ξεχάσει ένα εκ των δύο (* Η οριστική λύση για όλα αυτά τα προβλήματα είναι αυτή που προτείνεται παραπάνω, στο κεφάλαιο Φόρμες).

2. Λογικά σφάλματα

Και οι δύο τεχνικές χρησιμοποιούν την πολύ απλή εντολή `write` και μας οδηγούν κατευθείαν στο σημείο που υπάρχει το λάθος.

α) Επιτήρηση μεταβλητών

Σύμφωνα με τη μέθοδο αυτή τυπώνουμε όλες τις μεταβλητές που εμφανίζουν λάθος πριν και μετά από καίρια σημεία του προγράμματός μας. Μέχρις ενός σημείου τα αποτελέσματα θα συμφωνούν με αυτά που περιμέναμε. Από ένα σημείο και μετά όχι. Το λάθος βρίσκεται ανάμεσα.

Για παράδειγμα στο γνωστό πρόγραμμα που λέει να γίνει πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση όλων των κλασμάτων, έστω ότι στο τέλος μας εμφανίζει ακατανόητες απαντήσεις όπως άσχετους αριθμούς ή NaN (Not a number) κ.ο.κ.

Εισάγουμε πριν από την εκτέλεση των πράξεων δηλαδή αμέσως αφότου έχουμε συγκεντρώσει όλα τα δεδομένα τις εντολές :

```
write('Prin tin praksi. arithmitis1 : '); write(arithmitis1);
write('arithmitis2 : '); write(arithmitis2); …
```

κ.ο.κ. για όλες τις μεταβλητές που παρουσιάζουν πρόβλημα. Μετά την εκτέλεση των πράξεων ξαναεισάγουμε:

```
write('Meta tin praksi. arithmitis1 : '); write(arithmitis1);
write('arithmitis2 : '); write(arithmitis2); …
```

Με αυτόν τον τρόπο θα εμφανίσει το πρόγραμμα δύο επιπλέον γραμμές π.χ. :

```
Prin tin praksi. arithmitis1 : 8 arithmitis2 : -504
Meta tin praksi. arithmitis1 : 8 arithmitis2 : -504
```

Από τα παραπάνω γίνεται φανερό ότι σφάλμα βρίσκεται στο κομμάτι κώδικα που κάνει την εισαγωγή δεδομένων (με τα `readln`), αφού θεωρητικά ο `arithmitis2` θα έπρεπε να είναι θετικός.

Αν εμφάνιζε π.χ. τα εξής:

```
Prin tin praksi. arithmitis1 : 8 arithmitis2 : 4
Meta tin praksi. arithmitis1 : -32658 arithmitis2 : 4
```

τότε προφανώς το λάθος βρίσκεται στην εκτέλεση των πράξεων αφού κάπου μεταβάλλεται η τιμή του `arithmitis1` και δίνεται μία τιμή κάθε άλλο παρά αναμενόμενη.

Είναι πολύ σημαντικό να γνωρίζουμε σε ποιο σημείο του προγράμματος βρίσκεται το λογικό λάθος γιατί ελέγχοντας το συγκεκριμένο σημείο είναι σίγουρο ότι θα βρούμε το πρόβλημα.

β) Έλεγχος ροής

Εδώ εξετάζουμε χρησιμοποιώντας περίπου την ίδια μέθοδο μία διαφορετική κλάση προβλημάτων. Όταν χρησιμοποιούμε `if`, `while`, `for` και άλλες εντολές διακλάδωσης, επανάληψης ή αναδρομικές μεθόδους, τίθεται το πρόβλημα: Τι κάνει το πρόγραμμα μέχρι να τερματίσει ή γιατί δεν τερματίζει (κολλάει);

Για την απάντηση στα παραπάνω ερωτήματα χρησιμοποιούμε την εξής μέθοδο. Βάζουμε μία εντολή ελέγχου ροής δηλαδή μία `write('a'), write('b')...` με διαφορετικό κάθε φορά γράμμα σε τακτά διαστήματα μέσα στο πρόγραμμα. Εκτελώντας το πρόγραμμα εμφανίζονται τα γράμματα επιδεικνύοντας αναλυτικά τη ροή εκτέλεσης προγράμματος.

Για παράδειγμα έστω ότι θέλουμε να δούμε πως κινείται το παρακάτω σύνθετο πρόγραμμα:

```
program numeric_demo(input, output);
  var a,b,sum : integer;
begin
   sum:=10;
   for a:=0 to 10 do
     for b:= a to sum do
       while (a<sum) do
         if (a>b) then sum:=sum - 1
         else sum:=sum+1;
   writeln(sum);
end.
```

Δεν νομίζω ότι υπάρχει κανένας που θέλει να κάνει ανάλυση βήμα – βήμα αυτού του αλγορίθμου. Παρόλα αυτά είναι ενδιαφέρον να δει κανείς γιατί δίνει αποτέλεσμα `-32768`

Ας βάλουμε λοιπόν εντολές αναγνώρισης ροής σε όλα τα καίρια σημεία:

```
program numeric_demo(input,output);
  var a,b,sum : integer;
begin
   sum:=10;
   for a:=0 to 10 do
   begin

     write('a');
     for b:= a to sum do
     begin

       write('b');
       while (a<sum) do
       begin

         write('c');
         if (a>b) then
         begin

            write('d');
            sum:=sum – 1
```

```
         end
         else
         begin

            write('e');
            sum:=sum+1;

         end;
       end;
     end;
   end;
   writeln(sum);
end.
```

το αποτέλεσμα εκτέλεσης είναι το εξής:

```
abcecececececececececececececece ... cecebbbbbbbbbbaaaaaaaaaa-32768
```

Σε μια πρώτη ανάλυση βλέπουμε ότι δεν εμφανίζεται καθόλου το γράμμα d. Συνεπώς το a είναι πάντα μικρότερο του b. Μια δεύτερη παρατήρηση είναι ότι επαναλαμβάνεται το πρότυπο ce. Συνεπώς το a είναι μικρότερο του b όπως συμπεράναμε και παραπάνω και επιπλέον το a είναι μικρότερο του sum. Λόγο της επανάληψης του γράμματος e καταλαβαίνουμε ότι το sum αυξάνει κατά 1. Συνεπώς το sum αυξάνει συνεχώς, μέχρι που κάποια στιγμή ξεπερνά τα όρια του συστήματος για ακέραιους αριθμούς και γυρνά στο -32768 που είναι ο μικρότερος αρνητικός αριθμός. Αμέσως μόλις γίνει αυτό η συνθήκη του while δεν ικανοποιείται (a>sum) και το while τερματίζει δίνοντας τη σειρά του σε 10 εκτελέσεις του δεύτερου for όπως φαίνεται από τα 10 b. Μετά εκτελούνται 10 συνεχόμενες επαναλήψεις του πρώτου for όπως φαίνεται από τα 10 a και καμία του δεύτερου for αφού το sum είναι μικρότερο του a. Τέλος το πρόγραμμα τερματίζει εμφανίζοντας την τιμή του sum.

Η παραπάνω ανάλυση είναι πολύ πιθανόν να κούρασε όμως δείχνει το εξής. Με τη μέθοδο αυτή βρήκαμε όλη την πορεία του προγράμματος και ανιχνεύσαμε πολλά περίεργα φαινόμενα που συμβαίνουν κατά την εκτέλεσή του χωρίς να σκεφτούμε ούτε μία στιγμή θεωρητικά παρά μόνο εκτιμώντας τα αποτελέσματα τα οποία εμφάνισε στην οθόνη.

Καταλάβαμε στην ουσία τι κάνει το πρόγραμμα χωρίς να το διαβάσουμε.

Αυτή είναι μία πολύ δυνατή τεχνική με την οποία θα αντιμετωπίσετε και τα πιο σκληρά προβλήματα σε δύσκολους αλγορίθμους.

Δημήτρης Κουζής - Λουκάς 2002
users.ntua.gr/ge99149