Technical University of Crete

Εργαλεία Ανάπτυξης Λογισμικού και Προγραμματισμός Συστημάτων

ΠΛΗ211

Αναφορά Εργαστηρίου 1

Σπυριδάκης Χρήστος
ΑΜ: 2014030022

1.Εισαγωγή

Η υλοποίηση του κάθε ένα από τα δύο ζητούμενα προγράμματα του 1ου εργαστηρίου έχει παρόμοια δομή με το άλλο και αυτή αποτελείται από τρία βασικά σκέλη. Την αποδοχή επιλογών εισόδου του χρήστη, τον έλεγχο ορθότητας των αρχείων εισόδου καθώς και τελικά ο ζητούμενος υπολογισμός για το κάθε πρόγραμμα. Τα δύο πρώτα από αυτά θα εξηγηθούν στο ίδιο χωρίο της αναφοράς και για τα δύο scripts, αναφέροντας βέβαια τις όποιες διαφορές μεταξύ αυτών. Ενώ τέλος θα σχολιαστεί για το κάθε πρόγραμμα ξεχωριστά ο τρόπος υλοποίησης των βασικών υπολογισμών τους. Καθώς και πως ελέγχθηκε η λειτουργία των προγραμμάτων.

Για να περιγραφεί όσο πιο αναλυτικά γίνεται η υλοποίηση της άσκησης, η αναφορά έχει χωριστεί σε τρεις ενότητες:

1. Περιγραφή δομής προγραμμάτων
2. Εκτέλεση Υπολογισμών
3. Πειράματα

2.Περιγραφή

Algorithm 1 Regr Pseudocode

```
1:  procedure REGR(arguments)
2:      Read user input options
3:      Find out input files
4:      for Each input file do
5:          Check if file is valid and Calculate coefficients and squared error
6:          if Something is wrong then
7:              Print error message for this file
8:          else
9:              Print results for this file
10:     return 0
```

Algorithm 2 Results Pseudocode

```
1:  procedure RESULT(arguments)
2:      Read user input options
3:      Find out input files
4:      Check input files and then combine them
5:      If possible Calculate ranking based on combined data
6:      if Something is wrong then
7:          Print error message
8:      else
9:          Print results
10:     return 0
```

Image 2.1 - Basic structure of each script

Οι παραπάνω ψευδοκώδικες (**Image 2.1**) περιγράφουν σε γενικές γραμμές την μορφή τόσο του προγράμματος **regr** όσο και του **results**. Στην συνέχεια της αναφορά γίνεται μεγαλύτερη ανάλυση.

2.1 Input arguments

Όπως αναφέρθηκε ήδη και τα δύο προγράμματα υποστηρίζουν να δεχθούν παραμέτρους κατά την εκτέλεση τους. Για να γίνει αυτό χρησιμοποιήθηκε το εργαλείο getopts το οποίο ακολουθεί το standard POSIX.2 option syntax για τις εισόδους του. Δίνεται ιδιαίτερη προσοχή σε ποια option έπρεπε να ακολουθούνται από μία παράμετρο. Για περισσότερες πληροφορίες μπορείτε να διαβάσετε το help menu των εφαρμογών εκτελώντας:

Terminal - help menu display

```
$ bash regr -h          # Or ./regr -h, the same occurs for result script
```

```
while getopts ":c:de:ho:q" opt ; do
 case $opt in
   c)  CHECK_FILE="${OPTARG}";CHECK=${TRUE};check ${CHECK_FILE}; exit
$?;;
   d)  DEBUG=${TRUE} ;;
   e)  ERRORS_FILE="${OPTARG}" ;;
   h)  usage ; exit 0 ;;
   o)  OUTPUT_FILE="${OPTARG}" ;;
   q)  QUIET=${FALSE} ;;
   :)  error "Missing argument" ; exit 1  ;;
   \?) error "Invalid option"   ; exit 2  ;;
 esac
done
```

Image 2.1 - Option parsing

Μετά τα προαιρετικά flag - που χρησιμοποιήθηκαν κυρίως για θέματα αποσφαλμάτωσης, μπορούν να ακολουθήσουν όλα τα αρχεία εισόδου που θέλει να δώσει ο χρήστης. Τα οποία μετά τον απαραίτητο διαχωρισμό από τα υπόλοιπα arguments μεταβαίνουν ως είσοδο στην συνάρτηση ελέγχου ορθότητας των αρχείων εισόδου.

2.2 Check input files

Είναι σημαντικό πριν ξεκινήσουμε οποιαδήποτε ενέργεια να είμαστε σίγουροι ότι τα αρχεία που δίνονται ως είσοδος είναι πλήρως αποδεκτά. Ένα από τα πρώτα κομμάτια που πρέπει να ελεγχθούν είναι σχετικά με το ίδιο το αρχείο και όχι από το περιεχόμενο του. Οι πρώτοι έλεγχοι που γίνονται λοιπόν, είναι αν το αρχείο στο οποίο ζητούνται να κάνουμε τους

συγκεκριμένους υπολογισμούς, υπάρχει και είναι regular file, έχουμε δικαιώματα ανάγνωσης σε αυτό καθώς επίσης και ότι δεν είναι κενό καθώς έστω και ένα από τα παραπάνω να ισχύει δεν μπορούμε να κάνουμε κανένα υπολογισμό.
Αυτοί μόνο οι έλεγχοι όμως δεν αρκούν. Αν παραδείγματος χάρη, σε ένα αρχείο εισόδου για το **regr** για οποιοδήποτε λόγο σε μία από τις σειρές αντί για ένα δεκαδικό αριθμό υπάρχει ένα σύμβολο ή γράμμα, τότε στο πρόγραμμα θα προκληθεί μη αναμενόμενο σφάλμα. Για να λυθεί αυτό ελέγχεται με χρήση *regular expression* το κάθε αρχείο εισόδου. Στην **Image 2.2** εμφανίζεται η γενική ιδέα η οποία υπάρχει στον κώδικα. Όπου αποθηκεύουμε το καθαρό περιεχόμενο του αρχείου, αδιαφορώντας για τις κενές γραμμές, στην μεταβλητή **originalLines** ενώ στην μεταβλητή **goodLines** αποθηκεύουμε μόνο τις γραμμές οι οποίες ικανοποιούν το αποδεκτό format κάθε γραμμής που έχουμε ορίσει και στην συνέχεια συγκρίνουμε αν αυτά τα δύο είναι ίδια. Αν κάθε μη κενή γραμμή του αρχείου είναι στην μορφή που αναπαριστά το *regular expression* τότε πρέπει αυτές οι δύο μεταβλητές να ταυτίζονται. Σε αντίθετη περίπτωση τουλάχιστον μία από τις γραμμές έχει μη αποδεκτή μορφή.
Σε περίπτωση που ένα από τα παραπάνω καταπατείται το πρόγραμμα εμφανίζει σχετικό μήνυμα σφάλματος.

```
check(){
   ...
   if [ file_exists_is_regular_readable_not_empty ] ; then
      local originalLines=$(grep . ${inputFile})
      local goodLines=$(grep -E "${LINEREGEX}" ${inputFile})

      if [ "${originalLines}" == "${goodLines}" ] ; then
         result=0
      else
         result=11
      fi
   fi
   ...
}
```

Image 2.2 - Usage of regex to validate input file

2.2.1 Regular Expressions **regr**

Είναι βέβαια κατανοητό ότι το **LINEREGEX** για το **regr** πρόγραμμα δεν μπορεί να είναι το ίδιο με του **results** αφού είναι δύο τελείως ανεξάρτητα προγράμματα. Ο τρόπος δημιουργίας τους προέκυψε από τα δεδομένα της εκφώνησης. Για το **regr** η κάθε γραμμή ενός αρχείου είναι της μορφής:

num1:num2

Όπου num1 και num2 είναι δεκαδικοί αριθμοί. Από την στιγμή που δεν διευκρινίζεται περαιτέρω γίνονται αποδεκτοί κάθε ένας από τους τρόπους αναπαράστασης που εμφανίζονται στον **Table 2.2.1**. Που έπειτα όπου χρειάζεται γίνονται οι κατάλληλοι μετασχηματισμοί ώστε να μην υπάρχουν προβλήματα με την χρήση του εργαλείου **bc** που χρησιμοποιείται για τους υπολογισμούς.

12345,1	,2345	2.0	.1	-0.4	-.7	-3,5	-,93

Table 2.2.1 - Αποδεκτές αναπαραστάσεις δεκαδικών

Σύμφωνα με τα παραπάνω το regular expression για το **regr** script το οποίο δημιουργήθηκε και συμπεριλαμβάνει όλα τα παραπάνω είναι το εξής:

LINEREGEX="^[-]?([0-9]+[\.,]?|[0-9]*[\.,][0-9]+):[-]?([0-9]+[\.,]?|[0-9]*[\.,][0-9]+)$"

2.2.2 Regular Expressions **results**

Με όμοιο τρόπο δημιουργήθηκε το regular expression για το **results** με την παραδοχή ότι στα ονόματα των ομάδων δεν μπορεί να υπάρχει το σύμβολο της παύλας και της άνω-κάτω τελείας (όπως διευκρινήστηκε), καθώς είναι σημαντικά delimiter για τον διαχωρισμό των πεδίων. Επίσης στην εκφώνηση αναφέρεται ότι η κάθε γραμμή έχει την εξής μορφή χωρίς κανένα κενό:

Ομάδα1-Ομάδα2:Σκορ1-Σκορ2

Όμως αυτό αφορά τον διαχωρισμό μεταξύ των πεδίων, καθώς τα ονόματα των ομάδων μπορούν να περιλαμβάνουν ενδιάμεσα τους κενά. Συνεπώς καταλήγουμε ότι το όνομα θα πρέπει να είναι ένα αλφαριθμητικό τουλάχιστον ενός χαρακτήρα ώστε να έχει νόημα, όπου δεν πρέπει ούτε θα ξεκινάει ούτε θα τελειώνει με κενό σύμβολο και δεν πρέπει να να περιέχει παύλα και άνω-κάτω τελεία. Είναι σημαντικό να σημειωθεί ότι λόγω της ελευθερίας του συγκεκριμένο regular expression που χρησιμοποιήθηκε, γίνονται αποδεκτά ονόματα ομάδων σε διάφορες γλώσσες (λ.χ. την ελληνική, την κινεζική, κλπ).

LINEREGEX="^([^- :]|[^- :][^-:]*[^- :])-([^- :]|[^- :][^-:]*[^- :]):[0-9]+-[0-9]+$"

ΣΗΜΕΙΩΣΗ: Σε κανένα από τα δύο scripts δεν δίνεται κάποιο όριο στο μέγεθος των αριθμών εισόδου, συνεπώς θεωρήθηκε ότι μπορεί να είναι οποιοσδήποτε αριθμός που μπορεί να ικανοποιεί απλά το βασικό κριτήριο της εκφώνησης. Παραδείγματος χάρη, για το πρόγραμμα **results** γίνεται αποδεκτός οποιοσδήποτε θετικός ακέραιος αριθμός συμπεριλαμβανομένου του μηδέν.

3. Υπολογισμοί

Αφού ήμασταν σίγουροι ότι τα αρχεία εισόδου ήταν πλήρως αποδεκτά τότε μπορούσαμε να σκεφτούμε και να υλοποιήσουμε την μεθοδολογία με την οποία θα λύναμε το εκάστοτε πρόβλημα.

3.1 Regr

Για το πρόγραμμα **regr** όπως αναφέρεται και στην εκφώνηση γίνεται η εκτέλεση για το κάθε αρχείο μεμονωμένα. Συνεπώς για κάθε αρχείο διαβάζουμε τα διανύσματα X, Y και το μέγεθος αυτών και κάνουμε τους κατάλληλους υπολογισμούς. Στο **Image 3.1.1** φαίνεται με ποιον τρόπο γίνονται τα παραπάνω. Συγκεκριμένα σε κάθε ένα από τα τρια βήματα απλά χρησιμοποιούμε τα εργαλεία cat και grep, ώστε να πάρουμε από το αρχικό αρχείο όλο το περιεχόμενο χωρίς τις κενές γραμμές που μπορεί να υπάρχουν σε αυτό. Για το length απλά πρέπει να βρούμε το μήκος αυτών των γραμμών. Ενώ για τα vectors να χρησιμοποιήσουμε το κατάλληλο μέρος του αρχείο κάθε φορά. Ξέρουμε τι διαχωρίζει την μία πληροφορία από την άλλη συνεπώς μπορούμε να χρησιμοποιήσουμε το cut για να αποκόψουμε την πληροφορία που θέλουμε ανάλογα. Αφού μετατρέψουμε τους δεκαδικούς αριθμούς με την αναπαράσταση από κόμμα σε τελεία (για τους υπολογισμούς χρησιμοποιήθηκε το εργαλείο bc, στο οποίο δεν είναι αποδεκτή η χρήση δεκαδικών με χρήση του κόμματος) δημιουργούμε τα Array στα οποία αποθηκεύουμε τα Vectors.

```
length=$(cat "$1" | grep . | wc -l )
X_vector=$(cat "$1" | grep . | cut -d":" -f1 | tr "," "\.")
Y_vector=$(cat "$1" | grep . | cut -d":" -f2 | tr "," "\.")
unset X ; unset Y
declare -a X=(${X_vector}) ;
declare -a Y=(${Y_vector}) ;
```

Image 3.1.1 - Δημιουργία array X και Y

Πλέον είμαστε σε θέση να κάνουμε κάθε υπολογισμό. Στο **Image 3.1.2** εμφανίζεται ο τρόπος με τον οποίο υπολογίζονται τα απαραίτητα sum. Είναι σημαντικό εδώ να αναφερθεί ότι η μεταβλητή **DEC** έχει την τιμή 10, συνεπώς έχουμε 10 δεκαδικά ψηφία ακρίβεια στους ενδιάμεσους υπολογισμούς.

```
local sum_x=0 ; local sum_y=0 ; local sum_xy=0 ; local sum_x2=0
for ((i = 0; i <= ((${length} - 1)); i++)) ; do
        sum_x=$(echo "scale=${DEC}; ${sum_x} + ${X[i]}" | bc)
        sum_y=$(echo "scale=${DEC}; ${sum_y} + ${Y[i]}" | bc)
        sum_xy=$(echo "scale=${DEC}; ${sum_xy} + (${X[i]}*${Y[i]})" | bc)
        sum_x2=$(echo "scale=${DEC}; ${sum_x2} + (${X[i]}^2)" | bc)
done
```

Image 3.1.2 - Υπολογισμός των sum

Ένα από τα πιο σημαντικά σημεία της άσκησης είναι ο υπολογισμός των Linear Regression coefficients, συγκεκριμένα του **a**. Σε περίπτωση που έχουμε σταθερό διάνυσμα X τότε ο παρονομαστής για τον υπολογισμό του **a** μηδενίζεται συνεπώς αυτό είναι κάτι που πρέπει να προσέξουμε. Για να λυθεί αυτό, υπολογίζεται τόσο το dividend όσο και το divisor του **a**, ελέγχεται αν το divisor είναι διάφορο του μηδενός και μόνο τότε γίνονται οι υπόλοιποι υπολογισμοί. Σε αντίθετη περίπτωση εμφανίζεται απλά σχετικό μήνυμα στην έξοδο για το συγκεκριμένο αρχείο. Στην **Image 3.1.3** εμφανίζεται το χωρίο του κώδικα για το οποίο μόλις αναφερθήκαμε.

```
local a_dividend=$(echo "scale=${DEC}; ( (${length}*${sum_xy}) - (${sum_x}*${sum_y}) )" | bc)
local a_divisor=$(echo "scale=${DEC}; ( (${length}*${sum_x2}) - (${sum_x}*${sum_x}) )" | bc)
if [ "$(echo "${a_divisor} == 0" | bc -l)" -ne "1" ] ; then
    local a=$(echo "scale=${DEC}; ( ${a_dividend}/${a_divisor} )" | bc)
    local b=$(echo "scale=${DEC}; ( ${sum_y} - (${a}*${sum_x}) )/( ${length} )" | bc)

    local err=0
    for ((i = 0; i <= ((${length} - 1)); i++)) ; do
        err=$(echo "scale=${DEC}; ${err} + ( ${Y[i]} - (${a}*${X[i]} + ${b}) )^2 " | bc)
    done
else
    return 1
fi
```

Image 3.1.3 - Υπολογισμός `coefficients and standard squared error`

Τέλος αφού είχαμε τις ζητούμενες πληροφορίες χρειάστηκε απλά να τις εμφανίσουμε στην κατάλληλη μορφή. Για να το κάνουμε αυτό χρησιμοποιήθηκαν το εργαλείο awk με την χρήση της printf ώστε να εμφανίζονται μόνο τα 2 από τα 10 διαθέσιμα δεκαδικά ψηφία σε συνδυασμό με τη χρήση του Shell Parameter Expansion για τη διαγραφή του δεκαδικού μέλους αν αυτό δεν χρειάζεται.

```
a_p="$(echo ${a} | awk '{printf "%.2f", $0}')"
b_p="$(echo ${b} | awk '{printf "%.2f", $0}')"
err_p="$(echo ${err} | awk '{printf "%.2f", $0}')"
echo "FILE: $1, a=${a_p//\.00} b=${b_p//\.00} c=1 err=${err_p//\.00}"
```

Image 3.1.4 - Εμφάνιση αποτελεσμάτων

3.2 Results

Στο συγκεκριμένο πρόγραμμα δεν δόθηκε κάποια οδηγία να υποστηρίζεται η ταυτόχρονη εκτέλεση για πολλαπλά αρχεία εισόδου. Παρόλα αυτά πραγματοποιήθηκε, σε αυτήν όμως την περίπτωση το αποτέλεσμα είναι σε συνδυασμό των αρχείων αυτών.

Παράδειγμα.

```
$ ./results jan-2018.txt feb-2018.txt ...  # Υπολογίζει σε συνδυασμό των αρχείων εισόδου
```

Αφού έχει γίνει ο έλεγχος για κάθε αρχείο ξεχωριστά όπως προαναφέρθηκε στην **ενότητα 2.2** τα αρχεία αυτά συνενώνονται και κρατούνται μόνο οι μη κενές γραμμές σε μία μεταβλητή με όνομα **inptText**. Δημιουργούνται 4 arrays, ένα με numerical indexing και τα άλλα 3 είναι associative arrays (teamName, rankScore, scoredGoals και concededGoals). Σε αυτά θα αποθηκευτούν όλες οι πληροφορίες οι οποίες χρειαζόμαστε. Διαβάζουμε αρχικά μία-μία τις γραμμές του **inpText** και αποθηκεύουμε σε 4 προσωρινές μεταβλητές τις πληροφορίες που υπάρχουν σε κάθε γραμμή όπως φαίνεται στο **Image 3.2.1**. Σε αυτό το σημείο έπρεπε να συνυπολογιστούν όλες οι παραδοχές που έγιναν στο εδάφιο **2.2.2** σχετικά με τα ονόματα των ομάδων. Πριν γίνει η διευκρίνηση, είχε θεωρηθεί ότι τα ονόματα τον ομάδων θα μπορούσαν να περιέχουν την άνω-κάτω τελεία, για αυτό είναι περισσότερο πολύπλοκος από όσο ίσως χρειάζεται ο τρόπος που διαχωρίζουμε τις πληροφορίες. Παρόλα αυτά τελικά κρατήθηκε αυτός ο τρόπος υλοποίησης αν και λίγο πιο σύνθετος αφού καταλήγει στο ίδιο αποτέλεσμα χωρίς ιδιαίτερα αυξημένο κόστος. Επίσης ήταν σημαντικό, από την στιγμή που δεν απαγορεύεται βάση των requirements η ύπαρξη του star symbol, αυτό να μετασχηματιστεί σε ένα άλλο σύμβολο το οποίο θα γνωρίζαμε ότι δεν θα υπήρχε στα names (για να μπορεί να ανακτηθεί στο τέλος) διότι προκαλεί σε κάποιες περιπτώσεις προβλήματα στην χρήση των arrays. Κάναμε κάτι παρόμοιο και με τα κενά ώστε να έχουμε το κενό χαρακτήρα ως delimiter για το sorting.

```
teamA=$(echo ${line} | cut -d"-" -f1 | tr " " "-" | tr "*" ":")
teamB=$(echo ${line} | cut -d"-" -f2 | rev | cut -d":" -f2- | rev | tr " " "-" | tr "*" ":")
scoreA=$(echo ${line} | rev | cut -d":" -f1 | cut -d"-" -f2 | rev)
scoreB=$(echo ${line} | rev | cut -d"-" -f1 | rev)
```

Image 3.2.1 - Αποθήκευση σε προσωρινές μεταβλητές

Αφού έχουμε τις μεταβλητές γραμμής μπορούμε να υπολογίσουμε σιγά-σιγά, σύμφωνα με την εκφώνηση, τα περιεχόμενα των arrays. Αρχικά ελέγχουμε αν τα ονόματα των ομάδων

υπάρχουν ήδη στους πίνακες ή πρέπει να τα προσθέσουμε. Αφού γίνει αυτό βλέπουμε ποια ομάδα νίκησε στον κάθε αγώνα και ενημερώνουμε ανάλογα στον πίνακα rankScore στην θέση της εκάστοτε ομάδας τους βαθμούς της και τέλος ενημερώνουμε ανάλογα τα γκολ τα οποία σκόραρε όπως επίσης και αυτά που δέχτηκε. Στην **Image 3.2.2** φαίνεται η παραπάνω διαδικασία. Αξίζει να αναφερθεί ότι ο λόγος που χρησιμοποιείται το bc και όχι κάποιος από τους υπόλοιπους τρόπους πράξεων είναι διότι σε αυτό υπάρχει η δυνατότητα πραγματοποίησης πράξης και με πολύ μεγάλους αριθμούς (Ενότητα **2.2.2**).

ΣΗΜΕΙΩΣΗ: Τα ονόματα των ομάδων έχει θεωρηθεί ότι είναι case sensitive. Αυτό σημαίνει ότι το “Chelsea” είναι διαφορετικό από το “chelsea”. Ο λόγος που γίνεται αυτή η θεώρηση είναι για να υπάρχει συνοχή καθώς είναι εύκολο να πραγματοποιηθεί case insensitive για λατινικούς χαρακτήρες, αλλά είναι δύσκολο ταυτόχρονα να υποστηρίζεται για διάφορες γλώσσες. Ακόμα ένας λόγος είναι διότι πρακτικά υπάρχουν εναλλακτικές εκφωνήσεις των ονομάτων των ομάδων. Παραδείγματος χάρη η αναφορά “ΠΑΟ” ή “PAO” ή “Panathinaikos” για τον “Παναθηναϊκό”. Για να αποφευκτούν όλα αυτά θεωρήθηκε ότι η κάθε ομάδα στα αρχεία εισόδου πρέπει να λαμβάνει την επίσημη επιγραφή που έχει στο πρωτάθλημα για κάθε αγώνα.

```
exists "${teamsNames[@]}" "${teamA}"
if [ $? -eq 1 ] ; then
   teamsNames+=("${teamA}") ; rankScore+=(["${teamA}"]=0) ;
   scoredGoals+=(["${teamA}"]=0) ; concededGoals+=(["${teamA}"]=0) ;
fi
exists "${teamsNames[@]}" "${teamB}"
if [ $? -eq 1 ] ; then
   teamsNames+=("${teamB}") ; rankScore+=(["${teamB}"]=0) ;
   scoredGoals+=(["${teamB}"]=0) ; concededGoals+=(["${teamB}"]=0) ;
fi

if [ "$(echo "${scoreA} > ${scoreB}" | bc -l)" -eq "1" ] ; then                # Team A wins
    rankScore["${teamA}"]=$(echo "${rankScore["${teamA}"]} + 3" | bc)
elif [ "$(echo "${scoreA} < ${scoreB}" | bc -l)" -eq "1" ] ; then              # Team B wins
    rankScore["${teamB}"]=$(echo "${rankScore["${teamB}"]} + 3" | bc)
 else                                                                          # Draw
    rankScore["${teamA}"]=$(echo "${rankScore["${teamA}"]} + 1" | bc)
    rankScore["${teamB}"]=$(echo "${rankScore["${teamB}"]} + 1" | bc)
fi
scoredGoals["${teamA}"]=$(echo "${scoredGoals["${teamA}"]} + ${scoreA}" | bc)
scoredGoals["${teamB}"]=$(echo "${scoredGoals["${teamB}"]} + ${scoreB}" | bc)
concededGoals["${teamA}"]=$(echo "${concededGoals["${teamA}"]} + ${scoreB}" | bc)
concededGoals["${teamB}"]=$(echo "${concededGoals["${teamB}"]} + ${scoreA}" | bc)
```

Image 3.2.2 - Συμπλήρωση των array για κάθε γραμμή εισόδου

Μόλις ολοκληρωθεί αυτή η διαδικασία για κάθε γραμμή, υπάρχει στα associative arrays όλη η χρήσιμη πληροφορία που θέλουμε. Το μόνο που πρέπει να γίνει πλέον είναι να πραγματοποιηθεί το κατάλληλο sort και να εμφανιστεί. Στο **Image 3.2.3** μπορείτε να δείτε πώς το υλοποιούμε. Αρχικά δημιουργούμε ένα temporary file στο οποίο σε κάθε γραμμή του περιλαμβάνει όλες τις πληροφορίες εξόδου για κάθε ομάδα. Παρατηρήθηκε ότι υπήρξε πρόβλημα αν κάναμε αμέσως την ίδια διαδικασία με χρήση απλά μίας μεταβλητής για πολύ μεγάλα αρχεία, για αυτό τελικά χρησιμοποιήθηκε temp αρχείο. Έπειτα γίνεται αρχικά sort στο περιεχόμενο αυτού του αρχείου ως προς το rankScore (που είναι αριθμητικό σε ανάποδη σειρά ώστε να εμφανίζεται πρώτο το μεγαλύτερο σκορ) και έπειτα ως προς το teamNames και πριν εμφανιστεί αφαιρείται από τα ονόματα των ομάδων η παύλα που για ευκολία είχε προστεθεί νωρίτερα όπως επίσης γίνεται και η μετατροπή του star symbol.

```
echo -n "" > tmpFile.txt
for ((i=0;i < ${#teamsNames[@]};i++)) ; do
    line="${teamsNames[$i]} ${rankScore[${teamsNames[$i]}]} \
    ${scoredGoals[${teamsNames[$i]}]}-${concededGoals[${teamsNames[$i]}]}"
    echo ${line} >> tmpFile.txt
 done

echo "$(paste <( for ((i=1;i<=${#teamsNames[@]};i++)) ; do  echo "${i}." ; done ;) \
        <( cat tmpFile.txt | sort -t$' ' -k2,2nr -k1,1 | awk -F" " '{gsub("-", " ", $1); gsub(":", "*", $1);} \
        {print $1"\t"$2"\t"$3}') -d "\t")"
```

Image 3.2.3 - Δημιουργία εξόδου

---

4. Πειράματα

Αφού είχαν δημιουργηθεί και τα δύο script χρειάστηκε να γίνει έλεγχος σωστής λειτουργίας των προγραμμάτων. Πρώτοι έλεγχοι που έγιναν και στα δύο προγράμματα είναι σε ότι αφορά ολόκληρη την ενότητα **2.2**. Τι θα γίνει αν δοθεί ως είσοδο ένα μη regular αρχείο, το αρχείο εισόδου να μην υπάρχει ή να έχει λανθασμένο format κλπ. Ενδεικτικά εμφανίζεται η έξοδος σε κάποιες από αυτές τις περιπτώσεις για το πρόγραμμα regr. Παρόμοια ισχύουν για το results.

```
$ ./regr in1 in2 in3 in4 in5 in6
FILE: in1, there is not read permission
FILE: in2, file does not exist
FILE: in3, a=-0.15 b=2.57 c=1 err=199.50
FILE: in4, this is not a regular file
FILE: in5, file is empty
FILE: in6, something is wrong with the format of the file at any point
```

Image 4.1 - Regr παράδειγμα εκτέλεσης

4.1 Regr

Αφού είχε γίνει το παραπάνω, το πρώτο test το οποίο πραγματοποιήσαμε για αυτό το ερώτημα ήταν ο υπολογισμός των Linear Regression coefficients and standard squared error για ένα απλό αρχείο στο οποίο θα μπορούσαμε σε πρώτο επίπεδο να επαληθεύσουμε εύκολα την ορθότητα των πράξεων που γίνονται μέσα στο script. Ενδεικτικά θα δοθεί το παρακάτω παράδειγμα. Στην συγκεκριμένη περίπτωση τρέξαμε τον κώδικα δίνοντας του το όρισμα -d το οποίο ενεργοποιεί το debug mode, και μπορούμε να δούμε όλες τις ενδιάμεσες πληροφορίες που μας ενδιαφέρουν, ακόμα και τα vectors για τα οποία έτρεξε ο κώδικας.

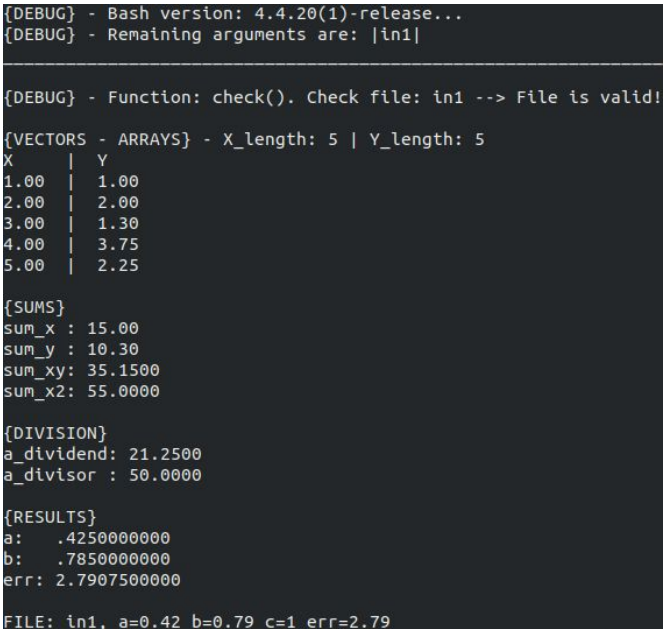


Image 4.1.1 - Απλό παράδειγμά εκτέλεσης regr

Δοκιμάσαμε επίσης τι θα συμβεί για σταθερό διάνυσμα X ώστε να δούμε σε αυτή την περίπτωση το πρόγραμμα αν θα λειτουργήσει σύμφωνα με τον σχεδιασμό.

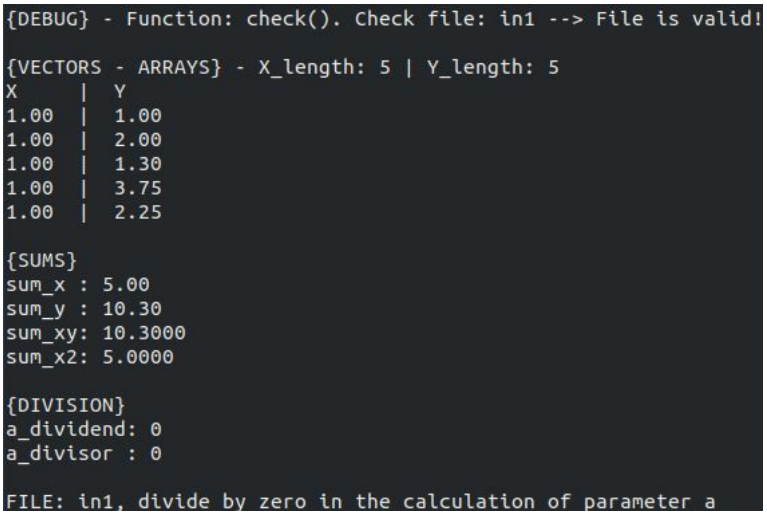


Image 4.1.2 - Εκτέλεση για σταθερό διάνυσμα X

Αφού είχαμε κάνει τα παραπάνω κάναμε μερικά πιο σύνθετα παραδείγματα. Στο πρώτο (παράδειγμα **4.1.1**) από αυτά που θα δοθούν, ακολουθήσαμε την reverse τακτική και βρήκαμε για διάφορες τιμές ενός vector X τις τιμές του vector Y για τις οποίες ικανοποιείται η σχέση (1) χωρίς να προσθέσουμε σε αυτές σφάλμα. Ενώ στο δεύτερο (παράδειγμα **4.1.2**) κάναμε ακριβώς το ίδιο για την σχέση (2) προσθέτοντας σε αυτές σφάλμα.

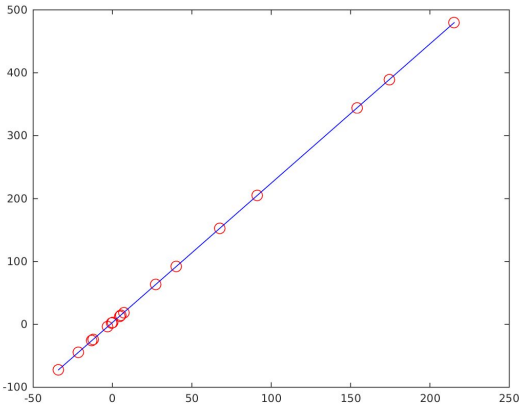
$$Y = 2.214 * X + 3.12 \quad (1)$$

$$Y = -0.451 * X - 23.84 \quad (2)$$

Παράδειγμα 4.1.1

```
4,5:13.083
7.124:18,892536
-21.4:-44,2596
-12.124:-23.722536
215.314:479.825196
27.1:63.1194
-13,168:-26.033952
154.12:344.34168
-.0:3.12
174,315:389.05341
5.12:14,45568
40.12:91.94568
-34.1:-72.3774
-3.12:-3.78768
67.4523:152.4593922
-,4:2.2344
91,224:205.089936
```

Περιεχόμενα αρχείου line



Γραφική αναπαράσταση των vectors του

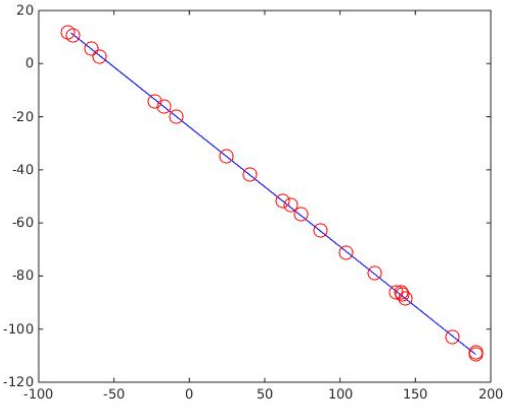
```
FILE: ../line, a=2.21 b=3.12 c=1 err=0
```

Αποτέλεσμα εξόδου για είσοδο το αρχείο line

Παράδειγμα 4.1.2

```
137,098929:-86.122617
123.147189:-78.928382
143.295666:-88.466345
24,752558:-35,003404
87.138318:-62.688381
74,092934:-56,804913
61.635007:-51.637388
-8.848110:-19,849502
-16,764753:-16.279096
39.872305:-41.822410
-22.921811:-14.404263
141.106630:-87.028090
190,284153:-108,756153
-77.280516:10.562513
67.166475:-53.230080
-59.787179:2.673018
140,485541:-86,296979
190.026016:-109,541733
-65.129274:5.533303
174.618242:-103.043827
-80,571107:11.595569
103.700869:-71.060092
```

Περιεχόμενα αρχείου witherror



Γραφική αναπαράσταση των vectors του

```
FILE: ../witherror, a=-0.45 b=-24 c=1 err=4.45
```

Αποτέλεσμα εξόδου για είσοδο το αρχείο witherror

Γνωρίζοντας ότι το πρόγραμμα λειτουργεί όπως θα περιμέναμε για μικρά αρχεία εισόδου, δημιουργήσαμε μία τυχαία είσοδο 1000 γραμμών η οποία προέκυψε από ένα generator ο οποίος δέχθηκε σαν είσοδο το regular expression - ελαφρός παραμετροποιημένο στο άνω όριο επανάληψης των αριθμών - του regr (ενότητα **2.2.1**), ώστε να επαληθεύσουμε ότι για κάθε αποδεκτή μορφή εισόδου δεν εμφανίζεται κάποιο μη αναμενόμενο πρόβλημα. Δεν γίνονται στιγμιότυπα εκτέλεσης από αυτά καθώς δεν έχουν κανένα λογικό χαρακτήρα, μπορεί όμως να θεωρηθεί μία μορφή crash test του κώδικα.

4.2 Results

Παρόμοια λογική ακολουθήθηκε και σε αυτό το πρόγραμμα. Αρχικά δοκιμάστηκε μία απλή είσοδος (παράδειγμα **4.2.1**) για την οποία ελέγθηκε σε πρώτη φάση η ομαλή λειτουργία του προγράμματος για τετριμμένες εισόδους. Στο παράδειγμα **4.2.2** δίνουμε τρία διαφορετικά αρχεία (έστω ότι είναι ένα μεγάλο με τα περιεχόμενα αυτών, αν δεν λάβουμε υπόψη την έξτρα λειτουργικότητα) τα οποία περιέχουν ονόματα ομάδων από διάφορες χώρες με διάφορους χαρακτήρες και κενά σε αυτά, σε αυτό το παράδειγμα ελέχθουμε την ορθή ταξινόμηση των ομάδων με βάση τα σωστά κριτήρια, καθώς για πιο απλές εισόδους μπορούμε να έχουμε σωστά αποτελέσματα μεν αλλά για λάθος λόγο. Στην συνέχεια (παράδειγμα **4.2.3**) δοκιμάστηκε το αρχείο που έχει ανέβει από συναδέλφους στη συζήτηση του courses σχετικά με αποτελέσματα πάνω σε πραγματικούς αγώνες. Ενώ τέλος όπως και για το regr προκειμένου να κάνουμε ένα ανάλογο crash test δημιουργήσαμε ένα αρχείο εισόδου 2000 γραμμών (με περίπου από όσο φάνηκε 1600 unic ονόματα ομάδων). Για το οποίο επαληθεύουμε για την ομάδα με όνομα “Η” τους βαθμούς και τα γκολ που είχαν στο πρωτάθλημα. Με την επαλήθευση του συγκεκριμένου ονόματος ομάδας μπορούμε να δούμε δύο πράγματα. Αρχικά ότι το πρόγραμμα δεν επηρεάζεται από ομάδες με ονόματα που είναι substring άλλων ομάδων και ότι βγάζει ακόμα και για αρκετά παράξενα αρχεία ή μεγάλα αρχεία εισόδου (παρόλα αυτά αποδεκτά) σωστά αποτελέσματα χωρίς εμφάνιση σφαλμάτων.

Παράδειγμα **4.2.1**

```
Bangladesh-Greece:1-2
Tanzania-Thailand:3-1
Benin-Bangladesh:2-4
Tanzania-Uruguay:2-6
Thailand-Kosovo:0-1
Uruguay-Rwanda:1-3
Australia-Philippines:2-2
Greece-Maldives:2-2
Maldives-Rwanda:2-2
```

Αρχείο εισόδου

```
1.      Greece  4       4-3
2.      Rwanda  4       5-3
3.      Bangladesh      3       5-4
4.      Kosovo  3       1-0
5.      Tanzania        3       5-7
6.      Uruguay 3       7-5
7.      Maldives        2       4-4
8.      Australia       1       2-2
9.      Philippines     1       2-2
10.     Benin   0       2-4
11.     Thailand        0       1-4
```

Αποτέλεσμα εκτέλεσης

Παράδειγμα **4.2.2**

```
Παναθηναϊκός-Bermuda:1-0
Άρης-Bermuda:1-0
Άρης-Α.Π.Σ. Ζάκυνθος 1961:1-0
Παναθηναϊκός-U.S. Virgin Islands:1-1
Α.Π.Σ. Ζάκυνθος 1961-Bermuda:1-0
Maldives-Bermuda:1-0
Bermuda-Maldives:0-1
广州富力-Παναθηναϊκός:0-1
Rwanda-Παναθηναϊκός:0-1
Maldives-Rwanda:1-0
British Virgin Islands-Maldives:0-1
Maldives-Άρης:1-0
广州富力-Maldives:0-1
Α.Π.Σ. Ζάκυνθος 1961-Maldives:0-1
Maldives-深圳FC:1-0
Maldives-广州富力:1-0
Α.Π.Σ. Ζάκυνθος 1961-深圳FC:1-0
广州富力-Α.Π.Σ. Ζάκυνθος 1961:0-1
```

Αρχείο εισόδου in1

```
广州富力-Άρης:0-1
Rwanda-Άρης:0-1
Άρης-Bermuda:1-1
Παναθηναϊκός-Maldives:0-1
Maldives-Άρης:1-0
Rwanda-Α.Π.Σ. Ζάκυνθος 1961:0-1
British Virgin Islands-Rwanda:1-1
Παναθηναϊκός-Maldives:1-0
Α.Π.Σ. Ζάκυνθος 1961-Curaçao:1-1
British Virgin Islands-Maldives:0-1
Άρης-Maldives:0-1
Maldives-British Virgin Islands:1-0
Άρης-Maldives:0-1
Maldives-Παναθηναϊκός:1-0
Maldives-Α.Π.Σ. Ζάκυνθος 1961:1-0
Maldives-Παναθηναϊκός:1-0
```

Αρχείο εισόδου in2

```
Maldives-British Virgin Islands:1-0
Maldives-广州富力:1-0
Α.Π.Σ. Ζάκυνθος 1961-Maldives:0-1
Maldives-Παναθηναϊκός:1-0
U.S. Virgin Islands-Maldives:0-1
Maldives-Παναθηναϊκός:1-0
Maldives-深圳FC:1-0
Παναθηναϊκός-Maldives:0-1
Rwanda-Maldives:0-1
British Virgin Islands-Maldives:0-1
Maldives-Rwanda:1-0
Maldives-Α.Π.Σ. Ζάκυνθος 1961:1-0
Rwanda-Maldives:0-1
Maldives-Α.Π.Σ. Ζάκυνθος 1961:1-0
Άρης-Maldives:0-1
Maldives-Bermuda:1-0
```

Αρχείο εισόδου in3

```
$ ./results in1 in2 in3
1.      Maldives        102     34-1
2.      Α.Π.Σ. Ζάκυνθος 1961    13      5-7
3.      Άρης    13      5-6
4.      Παναθηναϊκός    13      5-7
5.      Bermuda 1       1-7
6.      British Virgin Islands  1       1-6
7.      Curaçao 1       1-1
8.      Rwanda  1       1-8
9.      U.S. Virgin Islands     1       1-2
10.     广州富力        0       0-6
11.     深圳FC  0       0-3
```

Αποτελέσματα για την παραπάνω εκτέλεση

Όπως ήδη αναφέρθηκε μέσω αυτού του παραδείγματος ελέγχουμε τα εξής:

I. Γίνονται σωστά οι υπολογισμοί των σκορ και των γκολ
II. Ελέγχεται η δυνατότητα να έχουμε ονόματα ομάδων σε διαφορετικές γλώσσες
III. Ελέγχεται ότι μπορεί να γίνει η έξτρα λειτουργικότητα για πολλαπλά αρχεία χωρίς πρόβλημα
IV. Ελέγχεται ότι γίνεται σωστά η βασική ταξινόμηση βάση του σκορ συγκρίνοντας αριθμητικές τιμές. Εδώ είναι και ένα σημαντικό σημείο αν δεν το είχαμε προσέξει και ταξινούσαμε βάση αλφαβητικής θα έπρεπε οι ομάδες να είναι ταξινομημένες με βάση τα σκορ ως εξής: “13” > “102” > “1”. Όπου θα ήταν λάθος. Αντίθετα είναι ταξινομημένα με την σωστή σειρά.
V. Ελέγχεται η ταξινόμηση των ομάδων σε περίπτωση ισοβαθμίας, όπου για ομάδες με ονόματα στο ίδιο αλφάβητο είναι εύκολο να ελεγχθεί. Δεν σημαίνει όμως ότι είναι αυτονόητο και για ομάδες από διαφορετικά αλφάβητα. Χωρίς περισσότερη ανάλυση εξαιτίας της σωστής λειτουργίας για τις περιπτώσεις ταξινόμησης της ίδιας γλώσσας, θεωρείται ότι αρκεί η default λειτουργία του εργαλείου sort με τον τρόπο που χρησιμοποιήθηκε.

Παράδειγμα **4.2.3**

```
1.      Manchester City 98      95-23
2.      Liverpool       97      89-22
3.      Chelsea 72      63-39
4.      Tottenham       71      67-39
5.      Arsenal 70      73-51
6.      Manchester      66      65-54
7.      Wolvs   57      47-46
8.      Everton 54      54-46
9.      Leicster        52      51-48
10.     WestHam 52      52-55
11.     Watford 50      52-59
12.     Crystal 49      51-53
13.     Bournemouth     45      56-70
14.     Newcastle       45      42-48
15.     Burnley 40      45-68
16.     Southampton     39      45-65
17.     Brighton        36      35-60
18.     Cardiff 34      34-69
19.     Fulham  26      34-81
20.     Huddersfield    16      22-76
```

Αποτελέσματα προγράμματος για το παράδειγμα που ανέβασαν οι συνάδελφοι στο courses.

Όμιλος		Α	Ν	Ι	Η	ΓΥ	ΓΚ	ΓΔ	Β
1	Μάντσεστερ Σίτι	38	32	2	4	95	23	72	98
2	Λίβερπουλ	38	30	7	1	89	22	67	97
3	Τσέλσι	38	21	9	8	63	39	24	72
4	Τότεναμ	38	23	2	13	67	39	28	71
5	Άρσεναλ	38	21	7	10	73	51	22	70
6	Μάντσεστερ Γιουνάιτεντ	38	19	9	10	65	54	11	66
7	Γουλβς	38	16	9	13	47	46	1	57
8	Έβερτον	38	15	9	14	54	46	8	54
9	Λέστερ	38	15	7	16	51	48	3	52
10	Γουέστ Χαμ	38	15	7	16	52	55	-3	52
11	Γουότφορντ	38	14	8	16	52	59	-7	50
12	Κρίσταλ Πάλας	38	14	7	17	51	53	-2	49
13	Νιούκασλ Γιουνάιτεντ	38	12	9	17	42	48	-6	45
14	Μπόρνμουθ	38	13	6	19	56	70	-14	45
15	Μπέρνλεϊ	38	11	7	20	45	68	-23	40
16	Σαουθάμπτον	38	9	12	17	45	65	-20	39
17	Μπράιτον	38	9	9	20	35	60	-25	36
18	Κάρντιφ Σίτι	38	10	4	24	34	69	-35	34
19	Φούλαμ	38	7	5	26	34	81	-47	26
20	Χάντερσφιλντ	38	3	7	28	22	76	-54	16

Επίσημα αποτελέσματα αγώνων

Παράδειγμα **4.2.4**

```
15.     e       42      10015635581-3691859205
16.     H       42      12272218327-5471788401
17.     =       39      9121446554-18327673810
```

Image 4.2.4.1 - Σκορ ομάδας με όνομα Η

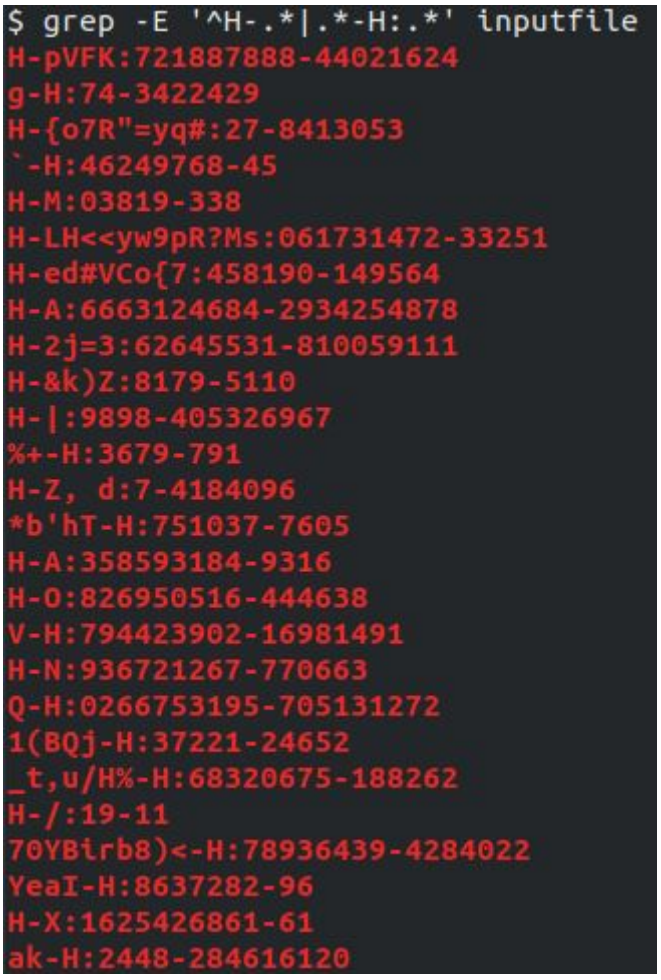


Image 4.2.4.2. - Αγώνες στους οποίους συμμετέχει η ομάδα Η

Σημείωση: Για να ελέγξουμε ότι η έξοδος είναι στην μορφή που ζητήθηκε με τα κατάλληλα tab τρέξαμε για μία τυχαία περίπτωση το εξής:

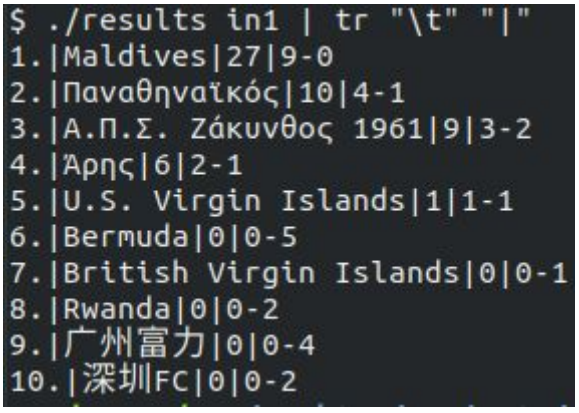


Image 4.3 - Έλεγχος δομής εξόδου