ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Λειτουργικά Συστήματα

**Ενότητα:** ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:04

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

**Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών**

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

# Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.


# Χρηματοδότηση


- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα του Πανεπιστημίου Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.




Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# Περιεχόμενα

# 1. Σκοπός της άσκησης

- Σενάρια Φλοιού Unix.
- Βρόχοι Επανάληψης.
- Αλλαγή ροής εκτέλεσης.
- Αριθμητικές πράξεις.
- Χειρισμός αλφαριθμητικών.
- Συνθήκες.
- Εξωτερικά προγράμματα.
- Grep.

# 2. Παραδοτέα

**(A)** 7 ερωτήσεις
**(C)** 9 ασκήσεις

# 3. Βρόχοι Επανάληψης

Σημαντικό στοιχείο των σεναρίων φλοιού είναι οι βρόχοι επανάληψης for. Με αυτόν τον τρόπο μπορούμε να επαναλάβουμε μια ομάδα από εντολές όσες φορές θελήσουμε. Η σύνταξη είναι:

```
for variable in …..
do
…
done
```

Με το παραπάνω κομμάτι κώδικα η for θα δίνει κάθε φορά στη μεταβλητή variable μια τιμή από το σετ που ακολουθεί και θα εκτελεστούν οι εντολές που βρίσκονται ανάμεσα στο do και done. Για παράδειγμα η εντολή:

```
for i in 1 2 3
do
echo hello $i
done
```

θα εμφανίσει

hello 1
hello 2
hello 3

Σε περίπτωση που θέλουμε να δημιουργήσουμε επαναλήψεις από κάποια αρχική τιμή, έως κάποια τελική τιμή με συγκεκριμένο βήμα θα πρέπει να χρησιμοποιήσουμε τη δομή **while**. Η σύνταξη της while είναι:

```
while [ expression ]
do
…
done
```

Η `expression` συντάσσεται με παρόμοιο τρόπο με τις συνθήκες if. Προκειμένου λοιπόν να δημιουργήσουμε το συγκεκριμένο βρόχο θα πρέπει:

- πριν από το while να αρχικοποιήσουμε τη μεταβλητή μας ( i=1),
- μέσα στην expression να θέσουμε τη συνθήκη που θέλουμε, όπως $i -lt 11 (δηλαδή να γίνεται η επανάληψη όσο η $i είναι μικρότερη από 11)
- μέσα στο βρόχο να έχουμε τη συνθήκη αύξησης της μεταβλητής **$i**. Αυτό επιτυγχάνεται με τη χρήση του εξωτερικού προγράμματος `expr` (**`man expr`**). Για να εκτελεστεί ένα εξωτερικό πρόγραμμα μέσα στο σενάριο φλοιού, απαιτείται η χρήση των ανάποδων εισαγωγικών `. Αν θέλουμε λοιπόν να γίνεται η αύξηση κατά 1 της μεταβλητής $i θα δώσουμε:
  **`i=`expr $i + 1``**
  (***προσοχή:*** *πριν και μετά το = δεν υπάρχουν κενά. Υπάρχουν κενά πριν και μετά το σύμβολο της πράξης)*

Το παρακάτω λοιπόν παράδειγμα:

```
#!/bin/sh
i=1
while [ "$i" -lt 10 ]
do
echo hello $i
i=`expr $i + 1`
done
```

θα εμφανίσει τα μηνύματα hello 1 έως hello 9 (αφού έχει το –lt δηλαδή “less-than” “μικρότερο”, ενώ αν ήταν –le δηλαδή “less-equal” “μικρότερο ή ίσο” θα εμφάνιζε τα μηνύματα hello 1 έως hello 10). Να επιβεβαιώσετε τα ανωτέρω με τη δημιουργία και εκτέλεση του script.

**(C1)** Δημιουργήστε το σενάριο φλοιού **c1.sh** το οποίο ζητάει από το χρήστη να πληκτρολογήσει το όνομά του *(με τη χρήση του read)* και τον αριθμό των επαναλήψεων που θα εκτυπωθεί το όνομά του. Στη συνέχεια εκτυπώνεται το όνομα που έδωσε ο χρήστης τόσες φορές όσες έχει ζητήσει.

## 4. Υλοποίηση αλλαγής ροής με break και continue

Αν θέλουμε να διακόψουμε την εκτέλεση ενός βρόχου μπορούμε να χρησιμοποιήσουμε την εντολή **break** για να διακόψουμε την εκτέλεση και να βγούμε έξω από το βρόχο.

Αν θέλουμε να μεταβούμε στην επόμενη εκτέλεση των εντολών του βρόχου, ακόμη και αν υπολείπονται εντολές για να εκτελεστούν στην τρέχουσα εκτέλεση του βρόχου, τότε χρησιμοποιούμε την εντολή **continue**.

Για παράδειγμα το script:

```
#!/bin/sh
a=0
while [ "$a" -le 10 ]
do
a=`expr $a + 1`
if [ $a -eq 2 -o $a -eq 4 ] ; then
continue
fi
if [ $a -eq 6 ] ; then
break

fi
echo -n "$a "
done
```

θα εκτυπώσει τις τιμές **1 3 5**. Αυτό θα γίνει γιατί όταν το `$a=2` ή *(-ο σημαίνει or)* `$a=4` τότε εκτελείται η εντολή `continue` και αμέσως πάμε στην επόμενη εκτέλεση του βρόχου παραλείποντας τις επόμενες εντολές, ενώ όταν `$a=6` τότε εκτελείται η εντολή `break` και σταματάει η εκτέλεση του πιο κοντινού βρόχου.

**(C2)** Δημιουργήστε το σενάριο φλοιού **c2.sh** το οποίο εμφανίζει κάποιους αριθμούς από 1 έως 100 με την ακόλουθη σειρά: 1 2 4 5 7 8 10 11....

*(TIP: θα χρησιμοποιήσετε την πράξη modulo του expr)*

**(A1)** Δημιουργήστε το αρχείο **a1-count2.sh** σύμφωνα με τον κώδικα που σας δίνεται παρακάτω. Παρατηρήστε ότι το σύμβολο **;** μπορεί να χρησιμοποιηθεί για να ενώσει στην ίδια γραμμή πολλαπλές εντολές. Κάποιες εντολές *(όπως το do και το then)* δε χρειάζονται να διαχωριστούν με **;** Το συγκεκριμένο πρόγραμμα θα τερματίσει κανονικά; Αν όχι που βρίσκεται το πρόβλημα και πως θα διορθωθεί;

---

```
#!/bin/sh
a=0 ; while [ "$a" -le 10 ] ; do
if [ $a -eq 2 -o $a -eq 4 ] ; then continue ; fi
if [ $a -eq 6 ] ; then break ; fi
a=`expr $a + 1` ; echo -n "$a " ; done
```

*****Δεν υπάρχει ετικέτα goto στα σενάρια φλοιού sh.**

## 5. Η δομή case

Μερικές φορές θέλουμε να χωρίζουμε την εκτέλεση ενός script σε τμήματα αναλόγως μιας παραμέτρου. Σε αυτές τις περιπτώσεις χρησιμοποιούμε τη δομή case η οποία έχει την παρακάτω σύνταξη:

```
case word in
pattern)
commands ;;
..
esac
```

όπου word είναι η μεταβλητή που εξετάζουμε, το pattern είτε το στοιχείο ακριβώς που θέλουμε είτε ένα pattern χρησιμοποιώντας τον χαρακτήρα ***** που είναι απεριόριστο πλήθος οποιονδήποτε χαρακτήρων. Για παράδειγμα:

```
#!/bin/sh
read number
case $number in
1)echo "You gave me 1" ;;
10*) echo "You have me big number" ;;
*)
echo "You gave me smt else"
;;
esac
```

Εκτελέστε το παραπάνω script και δοκιμάστε τις τιμές "`1 2 10 10aa 1010`" και παρατηρήστε τι εκτυπώνει *(δοκιμάστε το)*.

## 6. Αριθμητικές πράξεις στα σενάρια φλοιού

Αν θέλουμε να κάνουμε αριθμητικές πράξεις στις μεταβλητές *(π.χ. αύξηση της μεταβλητής κατά 1)* μπορούμε να χρησιμοποιήσουμε δύο τρόπους:

- Με τη χρήση του εξωτερικού προγράμματος `expr` , το οποίο υποστηρίζει αριθμητικές και λογικές πράξεις *(περισσότερα στο man expr)* και συντάσσεται ως εξής: **`expr στοιχείο1 πράξη στοιχείο2`** .
  Προσέξτε ότι τα στοιχείο1/στοιχείο2 μπορεί να είναι μεταβλητές ή αριθμοί. Για παράδειγμα **`expr $a + 1`** . Συνήθως το αποτέλεσμα του υπολογισμού τοποθετείται σε μια μεταβλητή, οπότε πρέπει να τοποθετήσουμε την εκτέλεση της expr μέσα σε ανάποδο μονό εισαγωγικό όπως: **a=`expr $a + 1`**

- Με τη χρήση της εσωτερικής εντολής εκτέλεσης αριθμητικών πράξεων δύο παρενθέσεων ως εξής: **`a=$(( $a + 1 ))`** . Προσέξτε ότι αμέσως μετά το = χρησιμοποιούμε το δολάριο $ και κολλητά δύο παρενθέσεις **((** . Η έκφραση έχει κενά από τις παρενθέσεις. Δώστε την εντολή **man sh** και πηγαίνετε στη σελίδα που αναφέρει τις αριθμητικές πράξεις που μπορούν να γίνουν με τη χρήση $$(expression) {Τίτλος Παραγράφου **Arithmetic Expansion**} .

Πόσες συνολικά πράξεις μπορούν να γίνουν:_____ **(Α2)**

**(C3)** Δημιουργήστε το σενάριο φλοιού **c3.sh** το οποίο ζητάει από το χρήστη να πληκτρολογήσει το όνομά του *(με τη χρήση της read)* και τον αριθμό των επαναλήψεων που θα εκτυπωθεί το όνομά του. Στη συνέχεια εκτυπώνεται το όνομα που έδωσε ο χρήστης τόσες φορές όσες έχει ζητήσει. Κάθε 10 φορές ερωτάται ο χρήστης αν θέλει να συνεχίσει και αν πατήσει Y ή y τότε συνεχίζεται η εκτύπωση, αν πατηθεί N ή n σταματάει, διαφορετικά επαναλαμβάνεται η ερώτηση.

**(C4)** Να κατασκευάσετε το script **c4.sh** το οποίο θα εκτυπώνει μια σειρά μηνυμάτων ως εξής. Θα ξεκινάει από μια τιμή που δίνει ο χρήστης και θα εκτυπώνει συνεχώς το αλφαριθμητικό “Στην αίθουσα υπάρχουν XX φοιτητές και θα φύγει ένας” ενώ μειώνεται η τιμή των φοιτητών. Μόλις φύγουν όλοι οι φοιτητές τότε εμφανίζεται το μήνυμα “Στην αίθουσα δεν υπάρχουν φοιτητές” και θα τελειώσει. Ο αριθμός των αρχικών φοιτητών θα δίνεται με δύο τρόπους. **(α)** είτε στη γραμμή εντολών ως 1η παράμετρος είτε **(β)** σε περίπτωση που δεν ανιχνευτεί παράμετρος στη γραμμή εντολών ή δε δοθεί αριθμός, τότε και μόνο τότε θα εμφανιστεί προτροπή προς το χρήστη και θα ζητηθεί αρχικός αριθμός φοιτητών.

Παράδειγμα εκτέλεσης:

```
Στην αίθουσα υπάρχουν 2 φοιτητές και θα φύγει ένας
Στην αίθουσα υπάρχουν 1 φοιτητές και θα φύγει ένας
Στην αίθουσα δεν υπάρχουν φοιτητές
```

## 7. Εκτέλεση εξωτερικών προγραμμάτων

Τις περισσότερες φορές θέλουμε στο φλοιό μας να εκτελούμε εξωτερικά προγράμματα, την έξοδο των οποίων να τοποθετούμε σε μεταβλητές. Όπως είδαμε αυτό γίνεται με το να τοποθετήσουμε την εντολή που θέλουμε να εκτελεστεί μέσα σε μονά ανάποδου εισαγωγικά `. Για παράδειγμα η παρακάτω εντολή υπολογίζει έναν τυχαίο αριθμό και τον τοποθετεί στη μεταβλητή random :

```
random=`od -vAn -N4 -tu4 < /dev/urandom`
```

Για να διαπιστώσετε το αποτέλεσμα της εντολής, δοκιμάστε να δώσετε κατευθείαν στο φλοιό την εντολή `od -vAn -N4 -tu4 < /dev/urandom` χωρίς εισαγωγικά και θα δείτε να σας εκτυπώνεται ένας αριθμός. Αν λοιπόν θέλουμε να τοποθετήσουμε τον τυχαίο αριθμό μέσα σε μια μεταβλητή τότε θα χρησιμοποιήσουμε τα μονά ανάποδα εισαγωγικά όπως προηγουμένως *(Για τους πιο περίεργους, αν δώσετε* `man od` *θα διαβάσετε τη σελίδα βοήθειας του εξωτερικού προγράμματος od καθώς και τη σημασία των παραμέτρων).*

**(C5)** Να κατασκευάσετε το script **c5.sh** το οποίο δημιουργεί μια τιμή με τυχαίο τρόπο *(χρησιμοποιώντας την προηγούμενη τεχνική)* και ζητάει από το χρήστη να πληκτρολογήσει μια τιμή. Αν η τιμή που πληκτρολογήσει ο χρήστης είναι ίδια τότε σταματάει η εκτέλεση και του εμφανίζει το μήνυμα "Συγχαρητήρια το βρήκες". Αν η τιμή που πληκτρολογεί ο χρήστης είναι μικρότερη, του εμφανίζει το μήνυμα "Έδωσες μικρή τιμή", διαφορετικά εμφανίζεται το μήνυμα "Έδωσες μεγαλύτερη τιμή". **ΠΡΟΑΙΡΕΤΙΚΑ:** Όταν ο χρήστης μαντέψει το μήνυμα και του εμφανιστεί το συγχαρητήρια, να εμφανίζεται και η προτροπή με τον αριθμό των προσπαθειών: "Προσπάθησες 15 φόρες".

## 8. Τμήματα αλφαριθμητικών με τη χρήση του cut

Αν θέλουμε να εμφανίσουμε ένα τμήμα ενός αλφαριθμητικού, τότε μπορούμε να χρησιμοποιήσουμε το εξωτερικό πρόγραμμα **cut**. Το cut δέχεται δεδομένα με τη χρήση της διασύνδεσης των εντολών δια μέσου της κάθετης γραμμής **|** .

Δηλαδή, `εντολή1 | εντολή2` . Στο παράδειγμα αυτό η εντολή1 στέλνει τα δεδομένα αντί στην οθόνη στην εντολή2. Αυτό δημιουργεί τη διαδιεργασιακή επικοινωνία με το όνομα "διασωλήνωση". Οι δυο πιο γνωστές λειτουργίες του cut είναι οι εξής:

- Μπορεί να εμφανίσει ένα τμήμα του αλφαριθμητικού από κάποιο συγκεκριμένο χαρακτήρα ως κάποιον άλλο. Η παράμετρος που χρησιμοποιούμε είναι η -b και μπορούμε είτε να δώσουμε εύρος χαρακτήρων ή συγκεκριμένους χαρακτήρες ή και τα δύο. Για παράδειγμα η εντολή:
  `echo "(unknown-unknown-FreeBSD)" | cut -b 1-3,10`
  θα εμφανίσει στην οθόνη τους χαρακτήρες 1-3 και τον χαρακτήρα 10, δηλαδή: **(unu**

- Μπορεί να εμφανίσει τμήματα ενός αλφαριθμητικού, που χωρίζονται με ένα συγκεκριμένο χαρακτήρα. Χρησιμοποιούμε την παράμετρο -f *(αριθμό πεδίου)* και -d χαρακτήρας διαχωριστικού. Για παράδειγμα η εντολή :
  `echo "(unknown1:unknown2:FreeBSD)" | cut -f 3 -d:`
  θα εκτυπώσει τη λέξη **FreeBSD)** .

- Μπορούμε να συνδυάσουμε εντολές cut με τη χρήση του συμβόλου | . Για παράδειγμα:
  `echo "(unknown1:unknown2:FreeBSD)" | cut -f 3 -d: | cut -b 1-4`
  θα εκτυπώσει τη λέξη **Free**.

- Υποστηρίζονται και άλλοι παράμετροι στη cut. Δείτε **man cut** για περισσότερες λεπτομέρειες.

**(A3)** Στη γραμμή εντολών δώστε **`echo $PATH`** η οποία εμφανίζει τη πλήρη διαδρομή αναζήτησης εκτελέσιμων προγραμμάτων. Να κάνετε copy-paste τη διαδρομή που εμφανίζεται.

**(A4)** Να δώσετε δύο παραδείγματα χρήσης της τμηματοποίησης αλφαριθμητικών με τη χρήση της παραμέτρου **`cut -b`** για τη μεταβλητή $PATH η οποία έχει ως αποτέλεσμα να απομονώνεται ένα path από όλα αυτά *(κάθε path διαχωρίζεται με τον χαρακτήρα : )*. Για παράδειγμα αν έχουμε τη διαδρομή /bin:/etc:/usr/bin: τότε θα πρέπει να βρω την εντολή για να εμφανιστεί είτε η /bin *(και μόνο αυτή)* είτε η /etc *(και μόνο αυτή)* είτε κάποια άλλη που υπάρχει.

**(A5)** Να δώσετε δύο παραδείγματα χρήσης της τμηματοποίησης αλφαριθμητικών με τη χρήση της παραμέτρου **`cut -f`** για τη μεταβλητή $PATH η οποία έχει ως αποτέλεσμα να απομονώνεται ένα path από όλα αυτά *(κάθε path διαχωρίζεται με τον χαρακτήρα : )*. Για παράδειγμα αν έχουμε τη διαδρομή /bin:/etc:/usr/bin: τότε θα πρέπει να βρω την εντολή για να εμφανιστεί είτε η /bin *(και μόνο αυτή)* είτε η /etc *(και μόνο αυτή)* είτε κάποια άλλη που υπάρχει.

## 9. Ανακατεύθυνση Εξόδου

Ως τώρα τα μηνύματα εμφανίζονται στην οθόνη που είναι η προεπιλεγμένη έξοδος. Αν θέλουμε να ανακατευθύνουμε την έξοδο σε κάποιο αρχείο τότε θα χρησιμοποιήσουμε το σύμβολο **>** ακολουθούμενο από το όνομα αρχείου. Για παράδειγμα δώστε:

```
ls -l /bin
```

και θα εμφανίσετε στην οθόνη τα αρχεία του καταλόγου /bin . Δώστε τώρα:

```
ls -l /bin > lsfile.txt
```

και δε θα εμφανιστεί τίποτα στην οθόνη. Θα τοποθετηθεί η έξοδος μέσα στο αρχείο lsfile.txt. Ανοίξτε το αρχείο lsfile.txt *(με την εντολή* **`more lsfile.txt`***)* και επιβεβαιώστε το.

## 10. Η εντολή exit

Αν θέλουμε να τερματίσουμε την εκτέλεση ενός φλοιού, τότε θα δώσουμε την εντολή exit. Η σύνταξή της είναι η:

```
exit exitcode
```

όπου **`exitcode`** είναι μια προαιρετική παράμετρος που αντιστοιχεί στον αριθμό EXIT_STATUS που θα επιστρέψει στο φλοιό. Η τιμή αυτή είναι η τιμή επιστροφής του προγράμματος. Ο αριθμός επιστροφής του τελευταίου προγράμματος τοποθετείται στην ειδική μεταβλητή $? . Για παράδειγμα, κατασκευάστε το σενάριο **testexit.sh** :

```
#!/bin/sh
exit 10
echo "Hello"
exit 20
```

κάντε το εκτελέσιμο με **chmod** και εκτελέστε το ως `./testexit.sh ; echo $?`

**(Α6)** Τι θα εκτυπωθεί με την εκτέλεση της εντολής `./testexit.sh ; echo $?` και γιατί;

## 11. Περισσότερα για συνθήκες (if)

Στο προηγούμενο εργαστήριο είδαμε κάποιες χρήσεις της if. Μπορούμε να τοποθετήσουμε επιπρόσθετα την εντολή **`elif`** *(=else if)* αν θέλουμε να εκτελεστεί κάποια ομάδα εντολών, αν ισχύει μια διαφορετική συνθήκη ή **`else`** για οποιαδήποτε άλλη περίπτωση. Η σύνταξη είναι:

```
if [ condition ] ; then
command1
elif [ condition ]; then
command2
else
command3
fi
```

Μπορούμε να κάνουμε έλεγχο ύπαρξης μεταβλητής με την παράμετρο `-z` . Δηλαδή, αν έχουμε τη συνθήκη `[ -z $variable ]` τότε αυτή θα επιστρέψει αληθή τιμή αν η μεταβλητή αυτή έχει κάποια στοιχεία.

Μπορούμε μέσα στις αγκύλες να τοποθετήσουμε περισσότερες εκφράσεις και να τις συνδέσουμε είτε με τη λογική πράξη AND είτε με τη λογική πράξη OR. Ο παρακάτω πίνακας εμφανίζει τους τρόπους με τους οποίους μπορούμε να το κάνουμε αυτό:

| Operation | Effect |
|---|---|
| [ ! EXPR ] | True if **EXPR** is false. |
| [ ( EXPR ) ] | Returns the value of **EXPR**. This may be used to override the normal precedence of operators. |
| [ EXPR1 -a EXPR2 ] | True if both **EXPR1** and **EXPR2** are true. |
| [ EXPR1 -o EXPR2 ] | True if either **EXPR1** or **EXPR2** is true. |

Οι πιο συχνά χρησιμοποιούμενες εκφράσεις χρησιμοποιούν τις λέξεις που εμφανίζονται στον παρακάτω πίνακα για τις αριθμητικές συγκρίσεις:

| | |
|---|---|
| [ ARG1 OP ARG2 ] | "OP" is one of `-eq`, `-ne`, `-lt`, `-le`, `-gt` or `-ge`. These arithmetic binary operators return true if "ARG1" is equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to "ARG2", respectively. "ARG1" and "ARG2" are integers. |

Εκτός από αριθμητικές συγκρίσεις μπορούν να γίνουν και συγκρίσεις αλφαριθμητικών, με τη χρήση του = . Για παράδειγμα, `"$var1" = "test"` . Ενώ με τη χρήση του != συγκρίνουμε αν είναι διαφορετικά, για παράδειγμα `"$var1" != "test"` .

Επίσης υπάρχει η ειδική μεταβλητή `$#` που είναι ο αριθμός των παραμέτρων γραμμής εντολών που έχει δώσει ο χρήστης.

**(C6)** Να κατασκευάσετε το αρχείο **c6.sh** το οποίο δέχεται μια παράμετρο και μόνο μια. Αν ο χρήστης δώσει μια παράμετρο του εμφανίζεται το μήνυμα “OK only 1”, αν δε δώσει κάποια παράμετρο του εμφανίζεται το μήνυμα “No parameter”, αν δώσει πολλαπλές παραμέτρους τότε εμφανίζεται το μήνυμα “Too many parameters”.

## 12. Περισσότερα για βρόχους επανάληψης

Ο βρόχος while μπορεί να χρησιμοποιηθεί εκτός από τη βασική σύνταξη για να διαβάζει γραμμές από ένα αρχείο και για κάθε γραμμή να εκτελείται με είσοδο αυτή τη γραμμή. Αυτό επιτυγχάνεται με δύο τρόπους:

- με το να δώσουμε είσοδο στο βρόχο με τη διασωλήνωση προγραμμάτων. Σύνταξη:

```
cat inputfile.txt | while read variable
do
…
done
```

- ή με το να δώσουμε είσοδο στο βρόχο από αρχείο άμεσα (**συνιστάται**). Σύνταξη:

```
while read variable
do
...

done < inputfile.txt
```

Με τις παραπάνω γραμμές και με τους δυο τρόπους διαβάζεται μια γραμμή από το αρχείο inputfile.txt και τοποθετείται στη μεταβλητή variable. Στη συνέχεια εκτελείται ο βρόχος με τις εντολές που βρίσκονται ανάμεσα στα do και done. Μόλις τελειώσει η εκτέλεση των εντολών, διαβάζεται η επόμενη γραμμή και τοποθετείται εκ νέου στη μεταβλητή variable και επαναλαμβάνεται η όλη διαδικασία. Όταν δεν υπάρχουν άλλες γραμμές, τότε η εκτέλεση του βρόχου σταματάει.

Μπορούμε μετά να χρησιμοποιήσουμε μέσα στο βρόχο την εντολή cut για να αναλύσουμε τη γραμμή. Για παράδειγμα:

```
cat inputfile.txt | while read var
do
tmp1=`echo $var | cut -b 1-4`
if [ "$tmp1" = "1024" ] ; then
echo "1024 detected at possition 1-4"
fi
done
```

Στο παραπάνω κομμάτι κώδικα μπορούμε να τοποθετήσουμε την cut μέσα στη συνθήκη ως εξής *(και θα έχει την ίδια λειτουργικότητα):*

```
if [ "`echo $var | cut -b 1-4`" = "1024" ] ; then
```

**(C7)** Να κατασκευάσετε το αρχείο **c7.sh** το οποίο θα δέχεται μια και μόνο μια παράμετρο από τη γραμμή εντολών. Η παράμετρος αυτή θα είναι όνομα αρχείου (ο έλεγχος ύπαρξης αρχείου γίνεται με το γράμμα `-e` στη συνθήκη if. Αν δεν υπάρχει αυτό το αρχείο να εμφανίζεται κατάλληλο μήνυμα και να τερματίζει το πρόγραμμα *(θα χρησιμοποιήσετε την if)*. Το σενάριο φλοιού θα ανοίγει το αρχείο και θα βρίσκει συνολικά πόσοι χρήστες έχουν σύνδεση 1024, πόσοι 2048 και πόσοι 8192. Το αρχείο που θα χρησιμοποιήσετε θα έχει γραμμές της μορφής είναι το:

```
John Papadakis/8192/Kavala
Peter O Toul/1024/Thessaloniki
John Pepas/512/Thessaloniki
Tasos K/8192/Kilkis
Jason F/512/Kozani
```

**(A7)** Χρησιμοποιώντας τη βοήθεια της ls βρείτε τι κάνει η παράμετρος -s. Εκτελέστε `ls -l /bin` και `ls -l -s /bin` (= ίδια εντολή με `ls -ls /bin`). Τι κάνει η παράμετρος `-s`;

**(C8)** Να κατασκευάσετε το αρχείο **c8.sh** το οποίο θα κάνει `ls -ls /bin` και θα προσθέτει όλους τους αριθμούς μεγέθους μπλοκ που βρίσκει. Στο τέλος θα αναγράφεται ο συνολικός αριθμός από μπλοκ. Να συγκριθεί με το μέγεθος που αναφέρει η εντολή `ls -ls /bin | more` στην πρώτη γραμμή. Είναι το ίδιο; Ναι/Όχι και γιατί.

## 13. Το πρόγραμμα grep (Α' μέρος)

Από **τα πιο σημαντικά** προγράμματα του UNIX είναι το grep *(και τα παράγωγα, όπως το egrep, pgrep, fgrep)*, το οποίο μπορεί και βρίσκει τις γραμμές που συμφωνούν με κάποια πρότυπα, που ονομάζονται κανονικές εκφράσεις και θα αναλυθούν σε επόμενο εργαστήριο. Για παράδειγμα αν θέλουμε να βρούμε τα αρχεία στον κατάλογο /bin που έχουν τους χαρακτήρες `name` στο όνομα, θα δώσουμε την εντολή *(δοκιμάστε το)*.

```
ls -l /bin | grep "name"
```

Αν θέλουμε να δούμε οπτικά σε ποιο σημείο βρέθηκε το συγκεκριμένο πρότυπο θα δώσουμε την παράμετρο `--color` ως εξής *(δοκιμάστε το)*:

```
ls -l /bin | grep --color "name"
```

Αν θέλουμε να μας μετρήσει πόσες γραμμές ταιριάζουν στο πρότυπο θα χρησιμοποιήσουμε το `-c` *(δοκιμάστε το)*:

```
ls -l /bin | grep -c "name"
```

**(C9)** Να κατασκευάσετε το αρχείο **c9.sh** το οποίο δέχεται μια παράμετρο από τη γραμμή εντολής. Η παράμετρος θα αντιστοιχεί σε κάποιον κατάλογο, π.χ. /bin θα κάνει `ls -l $1` και θα βρίσκει πόσες γραμμές έχουν τουλάχιστον ένα **a** και πόσες γραμμές έχουν τουλάχιστον ένα **e** και θα εκτυπώνει είτε το μήνυμα “more lines with e” είτε το μήνυμα “more lines with a” είτε το μήνυμα “same number of lines”. Μπορείτε αντί για a και e, να επιλέξετε οποιαδήποτε άλλα δυο γράμματα. Για παράδειγμα: `./c9.sh /bin`