

## ΕΡΓΑΣΤΗΡΙΟ 1:<sup>1</sup> Εισαγωγή, Χρήσιμες Εφαρμογές

Σκοπός του εργαστηρίου αυτού είναι η εξοικείωση με κάποιες εφαρμογές που θα μας φανούν πολύ χρήσιμες κατά τη διάρκεια του μαθήματος της Εισαγωγής στον Προγραμματισμό.

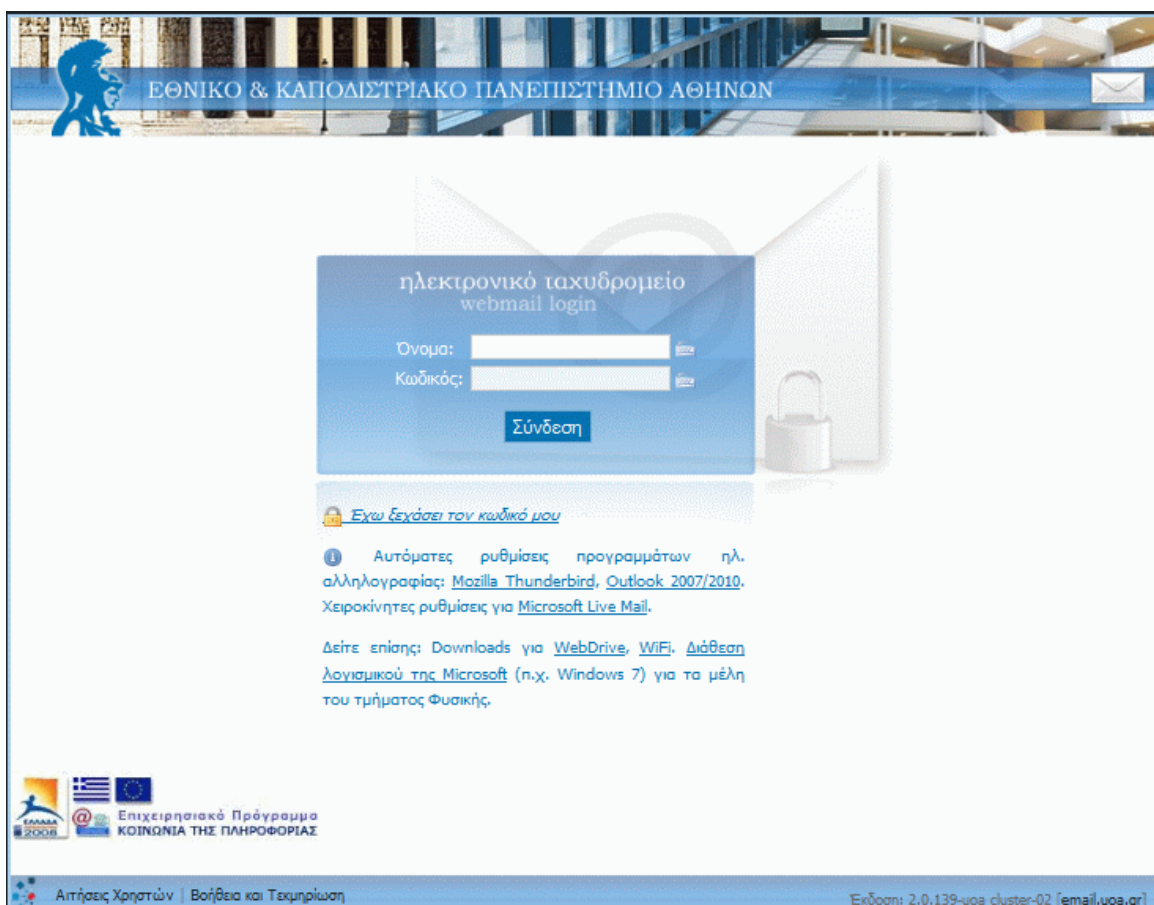
Για το λόγο αυτό θα μάθουμε:

- Να διαχειριζόμαστε την ηλεκτρονική μας αλληλογραφία μέσω της ιστοσελίδας διαχείρισης ηλεκτρονικής αλληλογραφίας webmail.
- Να εγγραφούμε στο ηλεκτρονικό φόρουμ του μαθήματος.
- Να συνδεόμαστε απομακρυσμένα στα μηχανήματα της σχολής μέσω του προγράμματος PuTTY.
- Να εκτελέσουμε τις πρώτες μας εντολές σε περιβάλλον Unix.

### 1. Το περιβάλλον διαχείρισης ηλεκτρονικής αλληλογραφίας webmail

Το webmail είναι ένα περιβάλλον διαχείρισης της ηλεκτρονικής μας αλληλογραφίας μέσω ιστοσελίδας για το e-mail που έχουμε από τη σχολή. Εδώ θα δούμε πως μπορούμε να στείλουμε ένα μήνυμα, να διαβάσουμε τα μηνύματα που λαμβάνουμε και να απαντήσουμε σε αυτά.

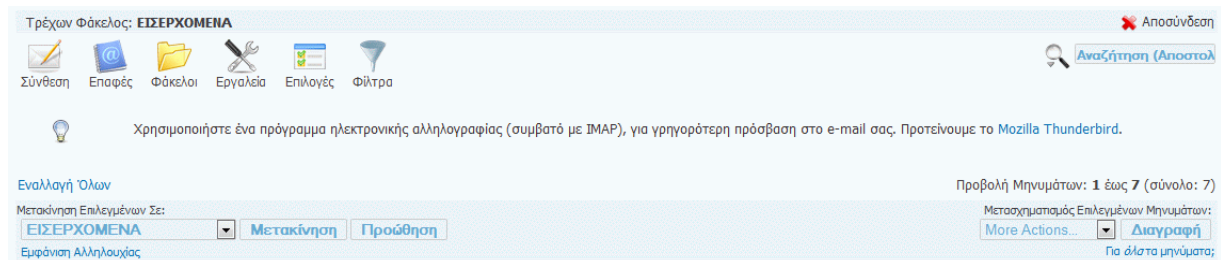
Ενώ είμαστε συνδεδεμένοι στο Internet, ανοίγουμε ένα παράθυρο φυλλομετρητή (browser) και πληκτρολογούμε την διεύθυνση <http://webmail.noc.uoa.gr>, οπότε και εμφανίζεται στην ιστοσελίδα η προτροπή για εισαγωγή των στοιχείων μας.



<sup>1</sup> Ευχαριστίες στους συνεργάτες του μαθήματος Δημήτρη Ψούνη, Στέφανο Σταμάτη, Νίκο Ποθητό, Μάνο Καρβούνη, Γιώργο Καστρίνη, Βασίλη Αναστασίου και στον Δρ. Ιωάννη Χαμόδρακα για τη συνεισφορά τους στη συγγραφή των εργαστηριακών φυλλαδίων του μαθήματος.

Πληκτρολογούμε το όνομα χρήστη sdiXXYYYYY και τον κωδικό μας και πατάμε το κουμπί «Σύνδεση».

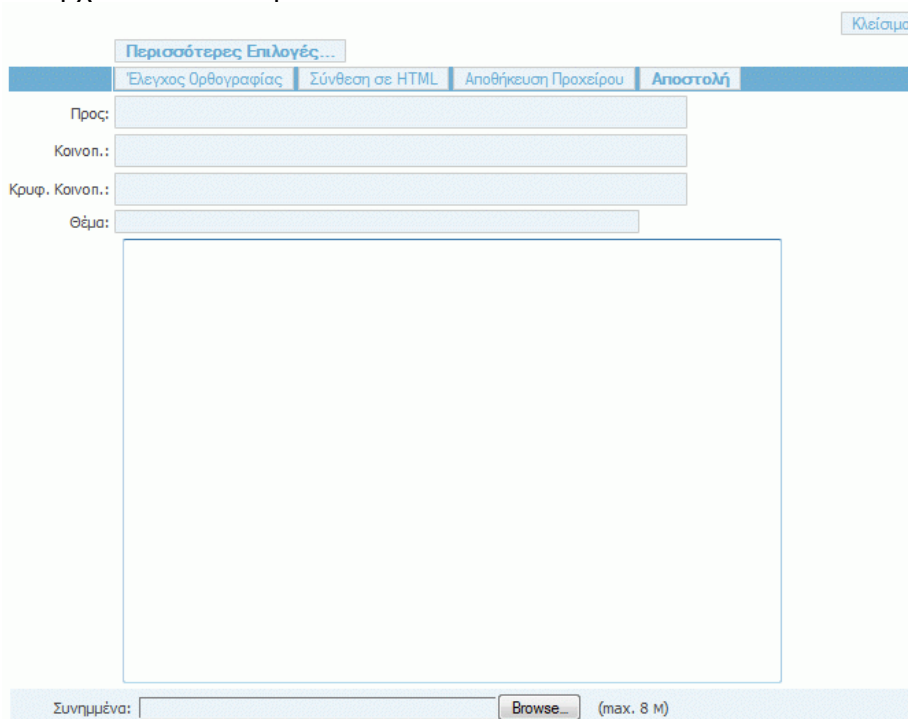
Στο πάνω μέρος της οθόνης εμφανίζεται ένα σύνολο από εικονίδια, που αντιστοιχούν στις διαθέσιμες επιλογές για την διαχείριση της ηλεκτρονικής μας αλληλογραφίας.



Στο κάτω μέρος της σελίδας υπάρχει μία λίστα με τα μηνύματα του ηλεκτρονικού μας ταχυδρομείου. Θα δούμε σε επόμενη ενότητα, πώς μπορούμε να τα διαχειριστούμε.

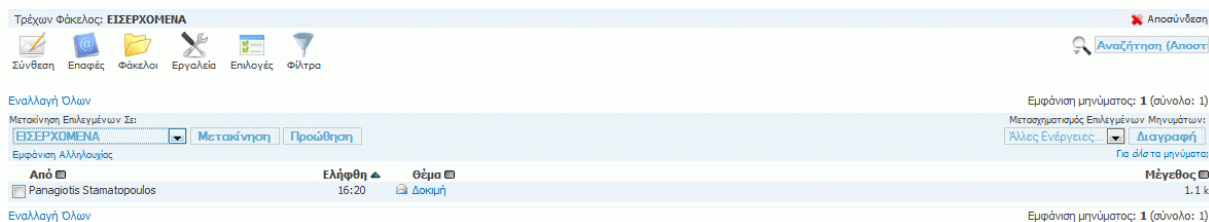
Με την επιλογή «Σύνθεση» μπορούμε να δημιουργήσουμε ένα νέο μήνυμα. Πατώντας το κουμπί εμφανίζεται ένα νέο παράθυρο (βλέπε επόμενη οθόνη) στο οποίο συμπληρώνουμε το μήνυμά μας:

- Προς: Συμπληρώνουμε την ηλεκτρονική διεύθυνση του αποδέκτη, ή τις ηλεκτρονικές διευθύνσεις χωρισμένες με κόμματα (εφόσον θέλουμε να το αποστείλουμε σε πολλαπλούς αποδέκτες).
- Κοινοπ.: Συμπληρώνουμε τις ηλεκτρονικές διευθύνσεις αυτών στους οποίους κοινοποιείται το μήνυμα.
- Κρυφ. Κοινοπ.: Συμπληρώνουμε τις ηλεκτρονικές διευθύνσεις αυτών που θέλουμε να λάβουν το μήνυμα χωρίς να εμφανίζονται οι διευθύνσεις τους σε αυτούς που λαμβάνουν το μήνυμα.
- Θέμα: Συμπληρώνουμε το θέμα του μηνύματος.
- Στο ορθογώνιο πλαίσιο συμπληρώνουμε το κείμενο του μηνύματος.
- Συνημμένα: Για την επισύναψη στο μήνυμα κάποιου αρχείου, κάνουμε κλικ στο “Browse” και επιλέγουμε το αρχείο που θέλουμε.

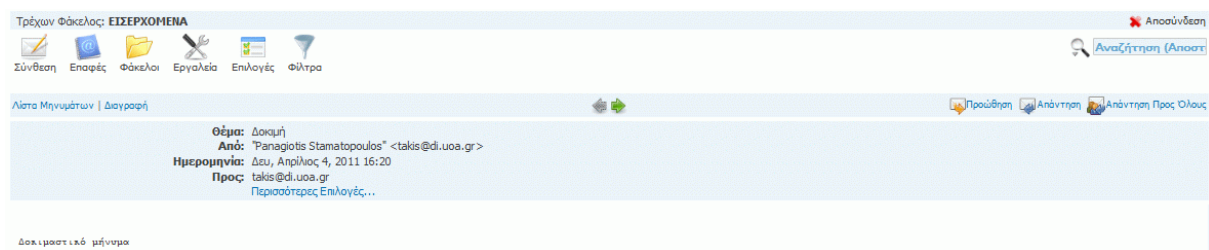


Αφού συμπληρώσουμε όσα από τα παραπάνω στοιχεία μας ενδιαφέρει, πατάμε το κουμπί «Αποστολή».

Ανάγνωση Εισερχόμενης Αλληλογραφίας



Για να διαβάσουμε ένα εισερχόμενο μήνυμα, κάνουμε κλικ στο θέμα του, οπότε και εμφανίζεται το μήνυμα σε αναλυτική μορφή.



Εδώ υπάρχουν οι διαθέσιμες επιλογές, από τις οποίες πιο ενδιαφέρουσες είναι οι εξής:

- «Απάντηση» με την οποία απαντάμε στο τρέχον μήνυμα. Εμφανίζεται μία οθόνη αντίστοιχη με αυτή της σύνθεσης νέου μηνύματος, μόνο που τα στοιχεία του παραλήπτη, του θέματος και του κειμένου του μηνύματος εμφανίζονται αρχικοποιημένα με τα στοιχεία του τρέχοντος μηνύματος.
- «Προώθηση» με την οποία προωθούμε το τρέχον μήνυμα σε άλλους παραλήπτες. Εμφανίζεται η οθόνη σύνθεσης μηνύματος, που επαναλαμβάνει το τρέχον μήνυμα, στην οποία πληκτρολογούμε τις ηλεκτρονικές διευθύνσεις των παραληπτών.
- «Διαγραφή» με την οποία διαγράφουμε το τρέχον μήνυμα και επαναφερόμαστε στην αρχική σελίδα με την εισερχόμενη αλληλογραφία.

Μόλις ολοκληρώσουμε τη διαχείριση της ηλεκτρονικής μας αλληλογραφίας, πατάμε το κουμπί «Αποσύνδεση» που βρίσκεται στο πάνω μέρος της οθόνης, ώστε να αποσυνδεθούμε από την εφαρμογή.

Εναλλακτικά, για να διαχειρίζεστε την ηλεκτρονική αλληλογραφία σας, μπορείτε να εγκαταστήσετε στον προσωπικό σας υπολογιστή ένα πρόγραμμα-πελάτη ηλεκτρονικής αλληλογραφίας (mail client), όπως, για παράδειγμα, το Thunderbird (<http://www.mozilla.org/el/thunderbird/>). Θα πρέπει στο πρόγραμμα αυτό να ορίσετε κάποιες παραμέτρους, ώστε να είναι σε θέση να διαχειρίζεται την ηλεκτρονική σας αλληλογραφία (παραλαβή και αποστολή μηνυμάτων). Αναλυτικές οδηγίες μπορείτε να βρείτε στον σύνδεσμο <http://www.noc.uoa.gr/hlektroniko-taxydromeio/ry8miseis.html>.

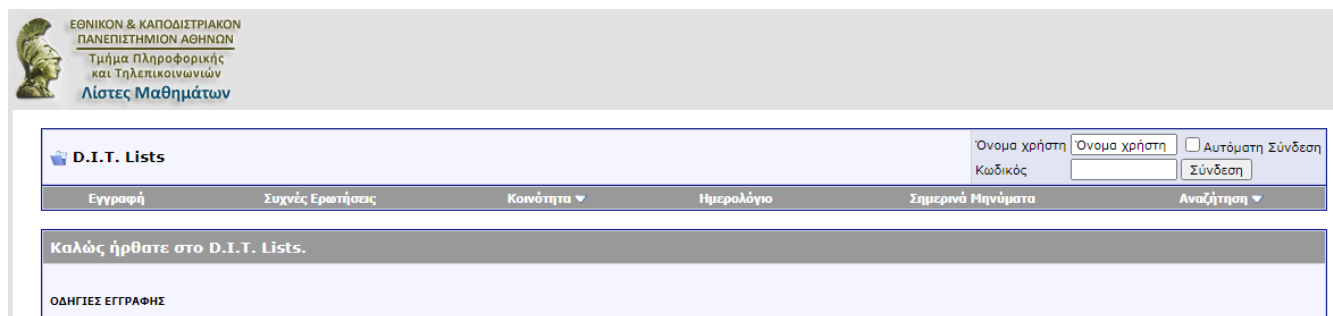
## 2. Εγγραφή στο Forum του μαθήματος

Όπως ήδη έχετε ενημερωθεί, στο μάθημα υπάρχει ηλεκτρονικό φόρουμ συζήτησης, μέσω του οποίου θα μπορούμε να ανταλλάσσουμε απόψεις για θέματα προγραμματισμού, για τις εργασίες του μαθήματος κ.λ.π.

Στην ενότητα αυτή θα δούμε πως μπορούμε να γραφτούμε στο φόρουμ του μαθήματος. Ανοίγουμε έναν browser και πληκτρολογούμε την ηλεκτρονική διεύθυνση:

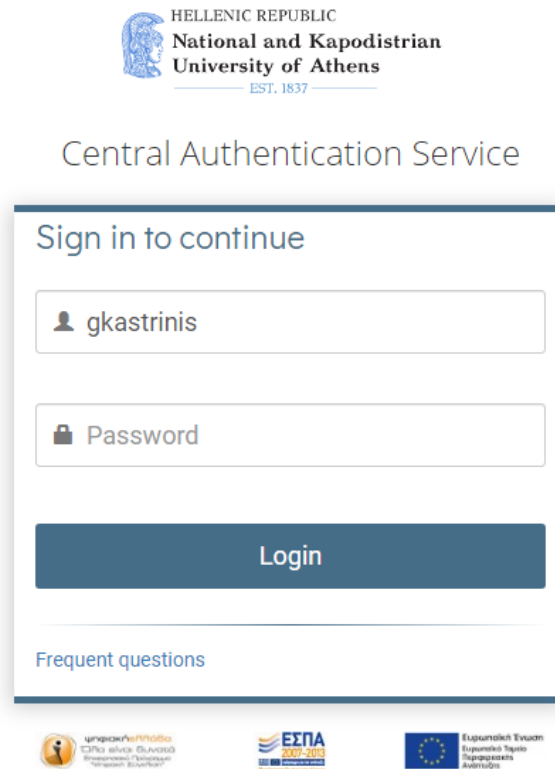
<https://lists.di.uoa.gr>

Στο μενού που εμφανίζεται πάνω αριστερά, επιλέγουμε «Εγγραφή» αφού διαβάσουμε τις σχετικές οδηγίες.



The screenshot shows the top part of the D.I.T. Lists website. On the left, there is a logo of the National and Kapodistrian University of Athens and the text: ΕΘΝΙΚΟΝ & ΚΑΠΟΔΙΣΤΡΙΑΚΟΝ ΠΑΝΕΠΙΣΤΗΜΙΟΝ ΑΘΗΝΩΝ, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Λίστες Μαθημάτων. On the right, there are input fields for 'Όνομα χρήστη' and 'Κωδικός', and a checkbox for 'Αυτόματη Σύνδεση'. Below these is a navigation menu with items: Εγγραφή, Συχνές Ερωτήσεις, Κοινότητα, Ημερολόγιο, Σημερινά Μηνύματα, and Αναζήτηση. A banner below the menu says 'Καλώς ήρθατε στο D.I.T. Lists.' and 'ΟΔΗΓΙΕΣ ΕΓΓΡΑΦΗΣ'.

Θα εμφανιστεί η κεντρική σελίδα πιστοποίησης του Πανεπιστημίου. Εκεί εισάγουμε τα στοιχεία του πανεπιστημιακού λογαριασμού μας:



The screenshot shows the Central Authentication Service login page. At the top, it says 'HELLENIC REPUBLIC National and Kapodistrian University of Athens EST. 1837'. Below that is the title 'Central Authentication Service'. The main content is a 'Sign in to continue' form with two input fields: one for the username 'gkastrinis' and one for the password. A blue 'Login' button is below the fields. At the bottom left of the form, there is a link for 'Frequent questions'. At the very bottom of the page, there are three logos: the National and Kapodistrian University of Athens logo, the ESPA logo, and the European Union logo.

Στη συνέχεια θα ανακατευθυνθούμε πάλι στην πλατφόρμα του lists, όπου και θα εμφανιστεί μία οθόνη με τους όρους χρήσης του φόρουμ, τους οποίους διαβάζουμε αναλυτικά:

Κανόνες του Forum

Πρέπει να συμφωνήσετε με τους παρακάτω κανόνες για να συνεχίσετε :

**Forum Rules**

Registration to this forum is free! We do insist that you abide by the rules and policies detailed below. If you agree to the terms, please check the 'I agree' checkbox and press the 'Register' button below. If you would like to cancel the registration, click [here](#) to return to the forums index.

**ΜΗΝ ΠΑΡΑΛΕΙΨΕΤΕ ΝΑ ΤΟ ΔΙΑΒΑΣΕΤΕ**

**[ΚΑΝΟΝΕΣ ΤΟΥ ΦΟΡΟΥΜ - ΔΙΑΒΑΣΤΕ ΟΠΩΣΔΗΠΟΤΕ](#)**

Για να γίνει δεκτή η εγγραφή σας...

Για να συνεχίσετε στο D.I.T. Lists forum, θα πρέπει να είστε κατόχος/κατόχος του πτυχίου

**Διάβασα και συμφωνώ στην τήρηση των κανόνων του D.I.T. Lists**

Πατώντας το κουτί επιλογής ότι διαβάσαμε και συμφωνούμε τους όρους χρήσης του forum και κάνοντας κλικ στο κουμπί «Εγγραφή», προχωράμε στην επόμενη οθόνη για να ολοκληρώσουμε την εγγραφή μας:

Προκειμένου να μπορείτε να δημοσιεύσετε μηνύματα στα D.I.T. Lists forums, πρέπει πρώτα να εγγραφείτε. Παρακαλούμε εισάγετε το επιθυμητό όνομα χρήστη, το email σας και τις άλλες απαιτούμενες πληροφορίες στην παρακάτω φόρμα.

**Όνομα χρήστη:** gkastrinis  
**Διεύθυνση Ηλεκτρονικού Ταχυδρομείου:** gkastrinis@di.uoa.gr

**Κωδικός**

Παρακαλώ εισάγετε έναν κωδικό για το λογαριασμό σας. Σας θυμίζουμε ότι οι κωδικοί είναι case-sensitive.

Κωδικός:  Επιβεβαίωση κωδικού πρόσβασης:

---

**Ζώνη Ώρας**

Τα πεδία ημερομηνίας και ώρας στο forum θα προσαρμοστούν αυτόματα για την περιοχή που βρίσκεστε. Διαλέξτε την κατάλληλη ζώνη ώρας απο την παρακάτω λίστα.

Ζώνη Ώρας:

Επιπρόσθετα, μπορείτε να ορίσετε τις κατάλληλες ρυθμίσεις για την αλλαγή θερινής/χειμερινής ώρας της περιοχής σας.

Επιλογές θερινής/χειμερινής ώρας:

**Αποδοχή Email**

Μερικές φορές οι administrators ίσως να θέλουν να σας στείλουν ανακοινώσεις με email.

Αν δεν θέλεις να λαμβάνεις αυτές τις ανακοινώσεις απενεργοποίησε αυτήν την επιλογή.

**Αποδοχή Email απο τούς Διαχειριστές**

Στον κωδικό εισάγουμε κάποιο λεκτικό της επιλογής μας. Αφού συμπληρώσουμε τα στοιχεία, πατάμε το κουμπί στο τέλος της σελίδας «Ολοκλήρωση Εγγραφής», οπότε μεταφερόμαστε στην ακόλουθη σελίδα που μας πληροφορεί ότι η διαδικασία εγγραφής στο φόρουμ ολοκληρώθηκε με επιτυχία:

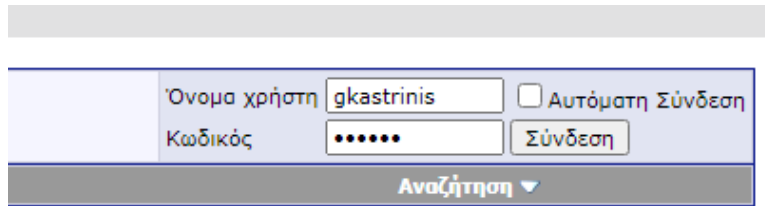
Μήνυμα vBulletin

Ευχαριστούμε, **gkastrinis**. Η εγγραφή σας έχει πλέον ολοκληρωθεί.

Τώρα μπορείτε να συνεχίσετε την επεξεργασία του **Προφίλ** σας, προκειμένου να καταχωρήσετε συμπληρωματικές πληροφορίες για σας, ή μπορείτε να ρυθμίσετε τις **Επιλογές** σας για να καθορίσετε τον τρόπο περιήγησης σε αυτή την ιστοσελίδα. Αν επιθυμείτε να κάνετε τα προαναφερόμενα αργότερα, μπορείτε να το κάνετε μέσω των επιλογών στον **Πίνακα Ελέγχου**.

Σε διαφορετική περίπτωση, μπορείτε απλά να επιστρέψετε στα **Forums** και να ξεκινήσετε τη δημιουργία θεμάτων..

Πλέον στην πάνω αριστερά γωνία μπορούμε να εισαγάγουμε τα στοιχεία μας, ώστε να συνδεθούμε με το σύστημα πατώντας το κουμπί «Σύνδεση».

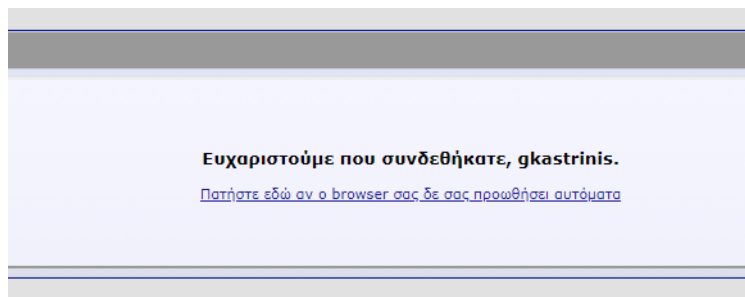


The image shows a login form with the following elements:

- Input field for 'Όνομα χρήστη' (Username) containing the text 'gkastrinis'.
- Input field for 'Κωδικός' (Password) containing six dots.
- Checkbox labeled 'Αυτόματη Σύνδεση' (Remember me).
- Button labeled 'Σύνδεση' (Login).
- Button labeled 'Αναζήτηση' (Search) with a dropdown arrow.

Αν συνδεόμαστε από τον προσωπικό μας υπολογιστή (π.χ. στο σπίτι μας), είναι καλή ιδέα να επιλέξουμε και το κουτί «Αυτόματη Σύνδεση», έτσι ώστε την επόμενη φορά που θα συνδεθούμε να μην χρειάζεται να πληκτρολογήσουμε ξανά το όνομα και τον κωδικό μας. Αν όμως συνδεόμαστε από δημόσιο υπολογιστή (π.χ. στη σχολή), να μην επιλέξουμε το κουτί αυτό, γιατί τότε ο επόμενος χρήστης που θα δουλέψει στο συγκεκριμένο υπολογιστή ενδέχεται να συνδεθεί στο φόρουμ με τα δικά μας στοιχεία, κάτι που, κατά πάσα πιθανότητα, δεν θέλουμε.

Αν πληκτρολογήσαμε σωστά τα στοιχεία μας, τότε εμφανίζεται το ακόλουθο επιβεβαιωτικό μήνυμα:

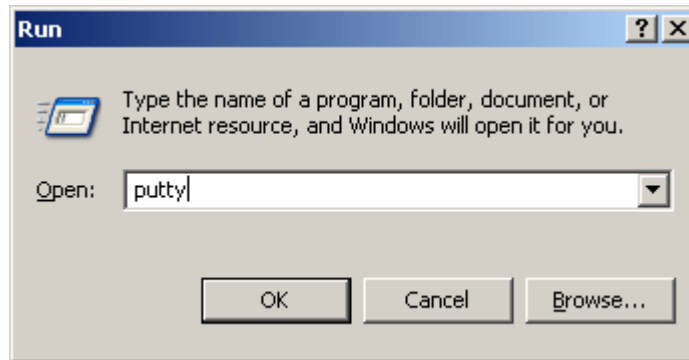


Πλέον είμαστε έτοιμοι να χρησιμοποιήσουμε το φόρουμ του μαθήματος.

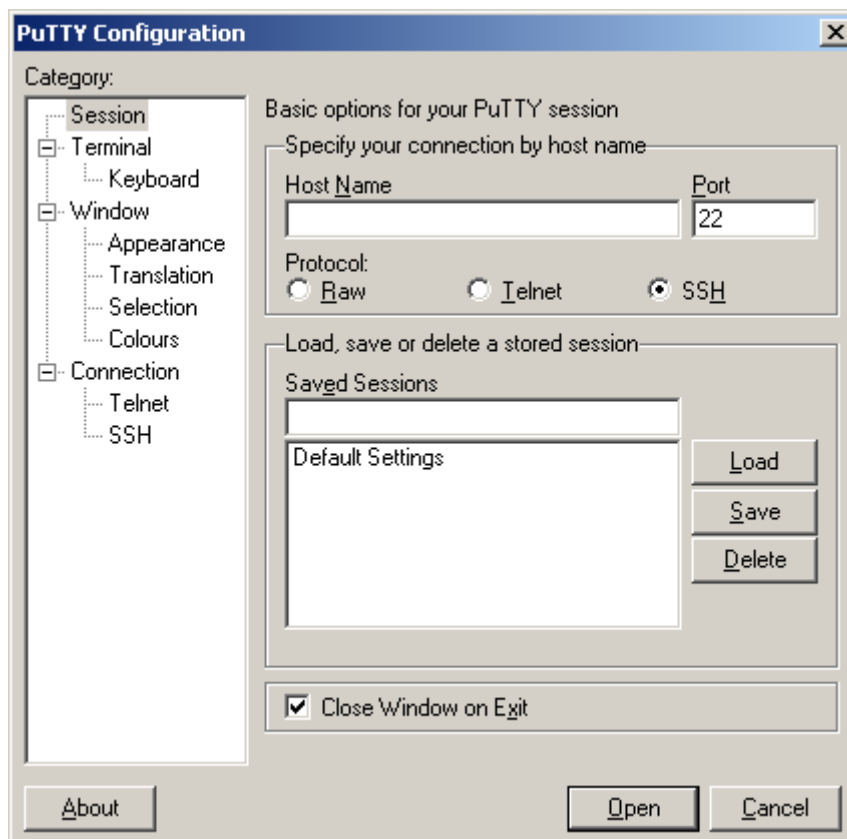
### 3. Η εφαρμογή PuTTY – Εξοικείωση με το Unix

Το PuTTY είναι πρόγραμμα απομακρυσμένης σύνδεσης, δηλαδή μέσω αυτού μπορούμε να συνδεόμαστε σε απομακρυσμένους υπολογιστές και να δουλεύουμε σαν να καθόμασταν μπροστά σε αυτούς! Έτσι, μπορούμε να συνδεθούμε και να δουλέψουμε στα συστήματα Linux της σχολής.

Πατάμε στα Windows<sup>2</sup>, Start->Run και στο παράθυρο που εμφανίζεται:



Γράφουμε “putty” και πατάμε OK. Η οθόνη που εμφανίζεται είναι η ακόλουθη:



---

<sup>2</sup> Μπορούμε να χρησιμοποιήσουμε το PuTTY και από τον υπολογιστή του σπιτιού μας, ώστε να συνδεόμαστε στους υπολογιστές της σχολής μέσω Internet. Θα πρέπει να κατεβάσουμε το εκτελέσιμο αρχείο putty.exe από την ηλεκτρονική διεύθυνση: <https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe>

Το σημαντικό κουτάκι είναι το «Host Name» στο οποίο συμπληρώνουμε το όνομα του υπολογιστή που θέλουμε να συνδεθούμε. Τα μηχανήματα που μπορούμε να συνδεθούμε έχουν ένα όνομα ακολουθούμενο από το .di.uoa.gr (το οποίο σημαίνει ότι “βρίσκονται” στη σχολή μας). Για τις ανάγκες του μαθήματος, μία λίστα με τους υπολογιστές που μπορούμε να χρησιμοποιήσουμε είναι η ακόλουθη:

- linux01.di.uoa.gr
- linux02.di.uoa.gr
- linux03.di.uoa.gr
- .....
- linux28.di.uoa.gr
- linux29.di.uoa.gr

Επιλέγουμε λοιπόν ένα από αυτά (π.χ. linux08.di.uoa.gr) και πατάμε το “Open”.



Γίνεται προτροπή να εισαγάγουμε το όνομα χρήστη μας (login as) όπου και πληκτρολογούμε το sdiXXXXYYY. Πατάμε Enter και βλέπουμε την προτροπή για εισαγωγή του κωδικού μας. Για λόγους ασφαλείας, όσο πληκτρολογούμε τον κωδικό μας, δεν εμφανίζεται κάτι στην οθόνη, οπότε μόλις το πληκτρολογήσουμε πατάμε Enter.

Αν όλα έχουν πάει καλά τότε θα δούμε στην οθόνη μας κάτι σαν το εξής:





```
linux08.di.uoa.gr - PuTTY
login as: ip
ip@linux08.di.uoa.gr's password:
linux08:/home/users/ip>
```

που σημαίνει ότι είμαστε στον κατάλογο που έχει τα αρχεία μας.

#### 4. Περιήγηση στο περιβάλλον του Unix

Το λειτουργικό σύστημα είναι τώρα έτοιμο να αλληλεπιδράσει μαζί μας, περιμένοντας τις εντολές μας για να δράσει αναλόγως.

Για το λόγο αυτό, πληκτρολογούμε στην γραμμή εντολών:

```
ls
```

Βλέπουμε τα περιεχόμενα του καταλόγου που στον οποίο βρισκόμαστε. Για να δούμε εκτενέστερες πληροφορίες για αυτά πληκτρολογούμε:

```
ls -l
```

Το αποτέλεσμα που θα δούμε στην οθόνη μας θα είναι κάτι σαν το εξής:

```
linux08:/home/users/ip>ls -l
total 5
drwx----- 2 ip dialout 512 Sep 16 2005 Mail/
drwx--x--x 2 ip dialout 512 Oct 15 2012 progs/
drwx--s--- 17 ip www 2048 Nov 27 11:26 public_html/
-rw----- 1 ip dialout 174 Nov 27 11:53 x-file.txt
linux08:/home/users/ip>
```

Ας δούμε λίγο πιο αναλυτικά τι σημαίνουν αυτά που βλέπουμε στην οθόνη μας:

- Το πρώτο γράμμα (d ή -) υποδηλώνει αν το αντικείμενο είναι κατάλογος ή αρχείο αντίστοιχα.
- Τα επόμενα 9 γράμματα ορίζουν τα δικαιώματα χρήσης του καταλόγου ή του αρχείου (θα επανέλθουμε σε αυτό σε επόμενο εργαστήριο).
- Ακολουθεί η πληροφορία του ιδιοκτήτη του αρχείου και η ομάδα στην οποία ανήκει.
- Το μέγεθος του.
- Η ημερομηνία και ώρα τελευταίας τροποποίησης.
- Το όνομα του αρχείου ή του καταλόγου αντίστοιχα.

Για να εισέλθουμε σε έναν κατάλογο πληκτρολογούμε:

```
cd ονομα_καταλόγου
```

Ας μπούμε τώρα στον κατάλογο Mail και να ελέγξουμε τα περιεχόμενα του. Πληκτρολογούμε:

```
cd Mail  
ls -l
```

Για να επιστρέψουμε στον αρχικό κατάλόγό μας, γράφουμε:

```
cd ..
```

Στο επόμενο εργαστήριο θα μάθουμε ένα υποσύνολο εντολών του Unix, που θα μας φανούν χρήσιμες για να μπορούμε να διαχειριζόμαστε τα αρχεία μας και να εκτελούμε ενέργειες επί αυτών, ώστε να είναι δυνατό να γράψουμε τα πρώτα μας προγράμματα σε γλώσσα C σε περιβάλλον Unix.

## 5. Ο κειμενογράφος pico (ή nano)

Εδώ θα φτιάξουμε ένα αρχείο κειμένου, θα γράψουμε κάτι σε αυτό και θα το αποθηκεύσουμε στον λογαριασμό μας. Το πρόγραμμα που θα χρησιμοποιήσουμε είναι ο κειμενογράφος pico.

Πληκτρολογούμε στο prompt

```
pico
```

Ανοίγει το περιβάλλον του pico, το οποίο φαίνεται στην ακόλουθη οθόνη:



Εδώ μπορούμε να πληκτρολογήσουμε κάποιο κείμενο και να το επεξεργαστούμε. Στο κάτω μέρος της οθόνης φαίνονται οι διαθέσιμες επιλογές που έχουμε, όπως για παράδειγμα να σώσουμε το κείμενο, να αναζητήσουμε σε αυτό, να βγούμε από το περιβάλλον του pico κ.λ.π.

Οι πιο ενδιαφέρουσες επιλογές είναι οι εξής:

Ctrl+O	Αποθήκευση Κειμένου. Εμφανίζει μία προτροπή για εισαγωγή του ονόματος του αρχείου
Ctrl+X	Έξοδος. Αν δεν έχουν αποθηκευτεί οι τελευταίες αλλαγές, τότε εμφανίζει μήνυμα για την αποθήκευση αυτών.
Ctrl+Y	Μετάβαση στην προηγούμενη σελίδα
Ctrl+V	Μετάβαση στην επόμενη σελίδα

Για παράδειγμα ας ακολουθήσουμε την διαδικασία για την αποθήκευση ενός μικρού κειμένου σε ένα αρχείο.

1. Πληκτρολογούμε ένα σύντομο κείμενο
2. Πατάμε Ctrl+O. Μας εμφανίζεται στο κάτω μέρος της οθόνης η προτροπή να δώσουμε ένα όνομα στο αρχείο που δημιουργήσαμε.

```
File Name to write : file.txt
^G Get Help      ^T To Files
^C Cancel        ^B Complete
```

3. Πληκτρολογούμε ένα όνομα (π.χ. file.txt) και πατάμε Enter.
4. Πατάμε Ctrl+X για να βγούμε από το περιβάλλον του pico.

Για να τυπώσουμε στην οθόνη τα περιεχόμενα του αρχείου που δημιουργήσαμε, πληκτρολογούμε:

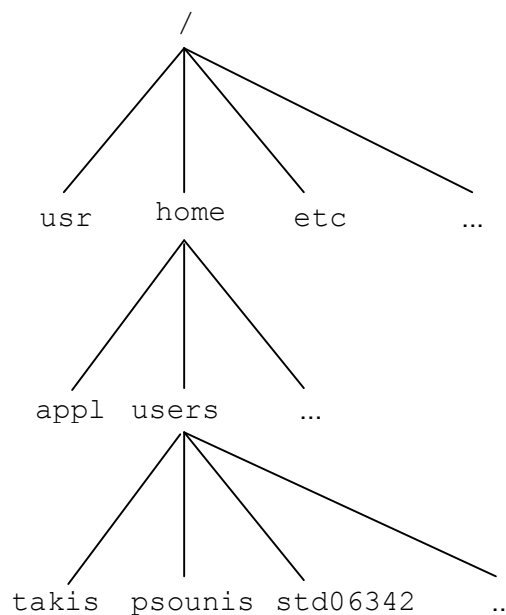
```
cat file.txt
```



## ΕΡΓΑΣΤΗΡΙΟ 2: Unix Tutorial

Σκοπός του εργαστηρίου αυτού είναι να έλθουμε σε επαφή με βασικές εντολές του Unix και την μεταγλώττιση προγραμμάτων C (με χρήση του μεταγλωττιστή `gcc`).

Για να οργανώσουμε τα αρχεία μας, τα τοποθετούμε σε καταλόγους. Κάθε κατάλογος μπορεί να περιέχει είτε αρχεία, είτε άλλους καταλόγους που τους λέμε και υποκαταλόγους του αρχικού. Όλα ξεκινούν από έναν αρχικό κατάλογο που λέγεται και ρίζα `/`. Για παράδειγμα, η δόμηση του συστήματος αρχείων στο μηχάνημα που έχετε συνδεθεί είναι ως εξής:



Όταν εργαζόμαστε σε ένα σύστημα, βρισκόμαστε σε έναν κατάλογο που θεωρείται ο τρέχων κατάλογος. Για παράδειγμα όταν συνδεόμαστε σε ένα μηχάνημα μέσω του Putty, αυτόματα βρισκόμαστε στον αρχικό μας κατάλογο: `/home/users/stdXXYYZ`. Για να αναφερθούμε τώρα σε κάποιον άλλο κατάλογο έχουμε δύο τρόπους:

- Είτε με απόλυτο μονοπάτι, ξεκινώντας από την ρίζα, για παράδειγμα: `/home/users/std06342`
- Είτε με σχετικό μονοπάτι, σε σχέση με τον τρέχοντα κατάλογο. Για παράδειγμα, αν ο τρέχων κατάλογος είναι ο `/home`, τότε σχετικό μονοπάτι σε αυτόν είναι το `users/std06342`. Επίσης σε σχέση με τον τρέχοντα κατάλογο, υπάρχουν και οι ειδικοί καταλόγοι:
  - `..` : Είναι συνώνυμο του γονικού καταλόγου (Για παράδειγμα, αν τρέχων κατάλογος είναι ο `/home/users/std06342`, τότε με `..` αναφερόμαστε στον κατάλογο `/home/users`).
  - `.` : Είναι συνώνυμο του τρέχοντος καταλόγου

Εδώ να τονίσουμε ότι τα ονόματα που χρησιμοποιούνται στους καταλόγους και στα αρχεία είναι *case-sensitive*, δηλαδή το όνομα καταλόγου «`Psounis`» είναι διαφορετικό από το όνομα καταλόγου «`psoUnis`». Αυτό είναι ένα γενικότερο χαρακτηριστικό του Unix, βέβαια. Όλα τα ονόματα που χρησιμοποιούμε, όχι μόνο για καταλόγους και αρχεία, αλλά και για εντολές του λειτουργικού ή εντολές μέσα σε διάφορα βοηθητικά προγράμματα (π.χ. κειμενογράφους) είναι *case-sensitive*.

Θα μελετήσουμε τώρα ένα πακέτο εντολών του Unix, που θα μας φανεί χρήσιμο για να διαχειριστούμε τα αρχεία μας, να γράψουμε προγράμματα C, να τα μεταγλωττίσουμε και να τα εκτελέσουμε.

Για τον λόγο αυτό θα διαβάζουμε την σύνταξη των εντολών<sup>1</sup> και θα τις χρησιμοποιούμε αναλόγως με το ζητούμενο στόχο. Ακολουθήστε βήμα-βήμα το παρακάτω σενάριο:

1. Εμφανίστε τα περιεχόμενα του καταλόγου `/usr/include`.

Η εντολή `ls` εμφανίζει τα περιεχόμενα καταλόγων.

Σύνταξη:

- `ls` : Εμφανίζει τα περιεχόμενα του τρέχοντος καταλόγου
- `ls ονομα_καταλόγου`: Εμφανίζει τα περιεχόμενα του καταλόγου με όνομα `ονομα_καταλόγου`, που έχει δοθεί είτε απόλυτα (από `root`) είτε σχετικά (από τρέχοντα κατάλογο)

Χρήσιμες επιλογές:

- `-a` : Εμφανίζει τα κρυφά αρχεία (αρχεία που αρχίζουν από τελεία)
- `-l` : Εμφανίζει εκτεταμένες πληροφορίες για τα περιεχόμενα του καταλόγου (δικαιώματα, μέγεθος, ημερομηνία τροποποίησης κ.λ.π.)

Χρήση μεταχαρακτήρων (ταιριάζουν τμήμα ονόματος αρχείου ή καταλόγου με τα διαθέσιμα)

- `*` : Ταιριάζει με κανέναν ή περισσότερους χαρακτήρες
- `?` : Ταιριάζει με ακριβώς έναν χαρακτήρα

Παραδείγματα χρήσης μεταχαρακτήρων:

- `ls *t` : Εμφάνισε τα αρχεία ή περιεχόμενα καταλόγων που το όνομα τους τελειώνει σε "t".
- `ls ??z8` : Εμφάνισε τα αρχεία ή περιεχόμενα καταλόγων που το όνομά τους έχει 4 γράμματα και τα δύο τελευταία είναι "z8"

2. Εμφανίστε το πλήρες όνομα του τρέχοντος καταλόγου.

Η εντολή `pwd` εμφανίζει το πλήρες όνομα του τρέχοντος καταλόγου.

3. Δημιουργήστε κάτω από τον τρέχοντα κατάλογο ένα νέο κατάλογο με όνομα `myWork`.

Η εντολή `mkdir` δημιουργεί τον κατάλογο που της δίνουμε σαν παράμετρο.

Συνταξη:

`mkdir ονομα_καταλόγου`

---

<sup>1</sup> Ένα ιδιαίτερα χρήσιμο κείμενο, με περαιτέρω τρόπους σύνταξης εντολών, πέρα από αυτές που θα αξιοποιήσουμε σήμερα, μπορούν να βρεθούν εδώ: <http://cgi.di.uoa.gr/~ip/Unix.pdf>

4. Μεταβείτε σε αυτόν τον νέο κατάλογο.

Η εντολή `cd` χρησιμοποιείται για την μετάβαση σε έναν κατάλογο.

Σύνταξη:

- `cd cat_name`: Μεταβαίνει στον κατάλογο `cat_name` κάτω από τον τρέχοντα κατάλογο
- `cd cat_path`: Μεταβαίνει στον κατάλογο που ορίζεται από το απόλυτο ή σχετικό μονοπάτι `cat_path`

Ειδική χρήση:

- `cd`: Μεταβαίνει στον αρχικό μας κατάλογο

5. Αντιγράψτε στον κατάλογο αυτό το πηγαίο αρχείο `~iphw/samples/mysin.c`

Η εντολή `cp` χρησιμοποιείται για την αντιγραφή αρχείων ή καταλόγων

Σύνταξη:

```
cp όνομα νέο_όνομα  
cp όνομα κατάλογος_προορισμού
```

Επιλογές:

- `-i`: Εμφανίζει ένα μήνυμα ελέγχου, όταν το αρχείο υπάρχει ήδη στον προορισμό και πρέπει να αντικατασταθεί.
- `-R`: Αντιγράφει τον κατάλογο `όνομα` μαζί με όλα τα περιεχόμενα του (αρχεία, υποκαταλόγους) σε κατάλογο με όνομα `νέο_όνομα`.

6. Δημιουργήστε το εκτελέσιμο αρχείο `mysin` από το πηγαίο αρχείο που μόλις αντιγράψατε.

Ο μεταγλωττιστής `gcc` χρησιμοποιείται για τη μεταγλώττιση πηγαίων αρχείων σε εκτελέσιμα.

Σύνταξη:

```
gcc -o εκτελέσιμο πηγαίο_αρχείο
```

Επιλογές:

- `-lm`: Ενσωμάτωση της μαθηματικής βιβλιοθήκης

7. Τρέξτε το εκτελέσιμο αυτό.

8. Αντιγράψτε στον τρέχοντα κατάλογο το πηγαίο αρχείο `~iphw/samples/mymain.c`

9. Αντιγράψτε στον τρέχοντα κατάλογο το αντικειμενικό αρχείο `~iphw/samples/myfunct.o`

10. Μεταγλωττίστε το πηγαίο αρχείο `mymain.c` στο αντίστοιχο αντικειμενικό αρχείο.

Ο μεταγλωττιστής `gcc` χρησιμοποιείται για την μεταγλώττιση πηγαίων αρχείων σε αντικειμενικά

Σύνταξη:

```
gcc -c πηγαίο_αρχείο.c : Παράγει το αντικειμενικό αρχείο με  
όνομα πηγαίο_αρχείο.o
```

11. Συνδέστε το αντικειμενικό αρχείο που κατασκευάσατε και το αντικειμενικό αρχείο `myfunct.o` δημιουργώντας το εκτελέσιμο αρχείο `myprog`.

Ο μεταγλωττιστής `gcc` χρησιμοποιείται για την σύνδεση αντικειμενικών αρχείων σε ένα εκτελέσιμο

Σύνταξη:

```
gcc -o εκτελέσιμο αντικ_αρχείο1 αντικ_αρχείο2 ....
```

12. Εκτελέστε το `myprog` και ακολουθήστε τις οδηγίες που θα εκτυπωθούν.

13. Διαγράψτε όλα τα αρχεία που δημιουργήσατε καθώς και τον κατάλογο `myWork`.

Η εντολή `rm` χρησιμοποιείται για τη διαγραφή αρχείων ή καταλόγων

Σύνταξη για διαγραφή αρχείων:

```
rm όνομα_αρχείου (χρήση και μεταχαρακτήρων)
```

Σύνταξη για διαγραφή καταλόγων (μαζί με όλα τα περιεχόμενά τους):

```
rm -r όνομα_καταλόγου
```

Επιλογές:

- `-i`: Εμφανίζει μήνυμα επιβεβαίωσης διαγραφής του αρχείου ή του καταλόγου.

14. Αντιγράψτε στον τρέχοντα κατάλογο το αρχείο `stdio.h` από τον κατάλογο `/usr/include`

15. Μετονομάστε το αρχείο που αντιγράψατε σε `Mystdio.h`

Η εντολή `mv` χρησιμοποιείται για τη μετονομασία ή μετακίνηση αρχείων ή καταλόγων

Σύνταξη:

```
mv αρχικό_όνομα τελικό_όνομα
```

```
mv αρχικό_όνομα καταλογος_προορισμού
```

Επιλογές:

- `-i`: Εμφανίζει μήνυμα ελέγχου εφόσον υπάρχει ήδη το αρχείο στον προορισμό και πρέπει να αντικατασταθεί.

16. Εμφανίστε στην οθόνη τα περιεχόμενα του αρχείου αυτού.



Η εντολή `cat` χρησιμοποιείται για την εμφάνιση των περιεχομένων αρχείων

Σύνταξη:

```
cat όνομα_αρχείου
```

Επιλογές:

- ο `-n` : Εμφανίζει αρίθμηση στις γραμμές του αρχείου

17. Δημιουργήστε στον τρέχοντα κατάλογο ένα κενό αρχείο με όνομα `.my_file` (προσέξτε την τελεία στην αρχή του ονόματος).

Η εντολή `touch` χρησιμοποιείται για τη δημιουργία κενών αρχείων.

Σύνταξη:

```
touch όνομα_αρχείου
```

18. Εμφανίστε τα δικαιώματα προστασίας του αρχείου αυτού και ό,τι άλλες χρήσιμες πληροφορίες μας δίνει το λειτουργικό σύστημα για αυτό.

19. Αλλάξτε τα δικαιώματα προστασίας έτσι ώστε εσείς (ο ιδιοκτήτης του) να έχει όλα τα δικαιώματα προστασίας (ανάγνωσης, εγγραφής και εκτέλεσης), τα μέλη της ομάδας στην οποία ανήκει το αρχείο να έχουν δικαιώματα ανάγνωσης και εκτέλεσης και οι υπόλοιποι να έχουν μόνο δικαίωμα ανάγνωσης.

Η εντολή `chmod` χρησιμοποιείται για τον καθορισμό των δικαιωμάτων προστασίας ενός αρχείου

Σύνταξη:

```
chmod XYZ όνομα_αρχείου
```

όπου X, Y, Z είναι οκταδικοί αριθμοί από το 0 έως το 7 που καθορίζουν τα δικαιώματα του ιδιοκτήτη, της ομάδας και των υπολοίπων αντίστοιχα.

Τα δικαιώματα προκύπτουν αν γράψουμε τον αριθμό σε δυαδική μορφή, όπου το 1<sup>ο</sup> bit καθορίζει το δικαίωμα ανάγνωσης, το 2<sup>ο</sup> bit το δικαίωμα εγγραφής και το 3<sup>ο</sup> bit το δικαίωμα εκτέλεσης.

20. Εμφανίστε πάλι τα δικαιώματα προστασίας του αρχείου `.my_file` (και τις άλλες πληροφορίες).

21. Εμφανίστε όλα τα αρχεία του τρέχοντος καταλόγου, μαζί με τα δικαιώματα προστασίας (και τις άλλες πληροφορίες), μη συμπεριλαμβανομένων των αρχείων που το όνομά τους αρχίζει από τελεία.

22. Το ίδιο με το προηγούμενο, αλλά να συμπεριληφθούν και τα αρχεία που το όνομά τους αρχίζει από τελεία.

23. Μέσω της εντολής `man` ενημερωθείτε σχετικά με τις δυνατότητες της εντολής `grep`.

Η εντολή `man` εμφανίζει πληροφορίες για την σύνταξη εντολών.

Σύνταξη:  
`man όνομα_εντολής`

24. Χρησιμοποιείστε την εντολή `grep` για να βρείτε μέσα στο αρχείο `/usr/include/stdlib.h` τις γραμμές που περιλαμβάνουν το κείμενο `size`.

25. Επαναλάβετε την προηγούμενη εντολή αλλά να κάνετε ανακατεύθυνση της εξόδου της στο αρχείο `grepout.txt`

Η ανακατεύθυνση εξόδου σε αρχείο γίνεται με την ακόλουθη σύνταξη:

Εντολή > ονομα\_αρχείου

26. Συνδυάστε με κάποιον τρόπο τις εντολές `ls` και `grep` για να βρείτε στον τρέχοντα κατάλογο αρχεία στα οποία ο ιδιοκτήτης έχει δικαιώματα ανάγνωσης και εγγραφής, αλλά όχι εκτέλεσης και όλοι οι υπόλοιποι δεν έχουν κανένα δικαίωμα.

Η σωλήνωση της εξόδου ενός προγράμματος στην είσοδο ενός άλλου, γίνεται με την ακόλουθη σύνταξη:

Πρόγραμμα | Πρόγραμμα

27. Αντιγράψτε στον τρέχοντα κατάλογο το πηγαίο αρχείο `~iphw/samples/capitalize.c` δημιουργήστε το αντίστοιχο εκτελέσιμο και τρέξτε το με ανακατεύθυνση της εισόδου από το ίδιο το πηγαίο αρχείο `capitalize.c` στέλνοντας την έξοδο (πάλι με ανακατεύθυνση) στο αρχείο `CAPITALIZE.c`

Η ανακατεύθυνση εισόδου και εξόδου γίνεται με την ακόλουθη σύνταξη:

Εντολή < Αρχείο\_για\_είσοδο > Αρχείο\_για\_έξοδο

28. Δοκιμάστε να μεταγλωττίσετε το αρχείο `CAPITALIZE.c`. Μην ανησυχείτε όμως αν δεν μπορεί να μεταγλωττιστεί (γιατί άραγε;)

## ΕΡΓΑΣΤΗΡΙΟ 3: Προγραμματιστικά Περιβάλλοντα και το Πρώτο Πρόγραμμα C

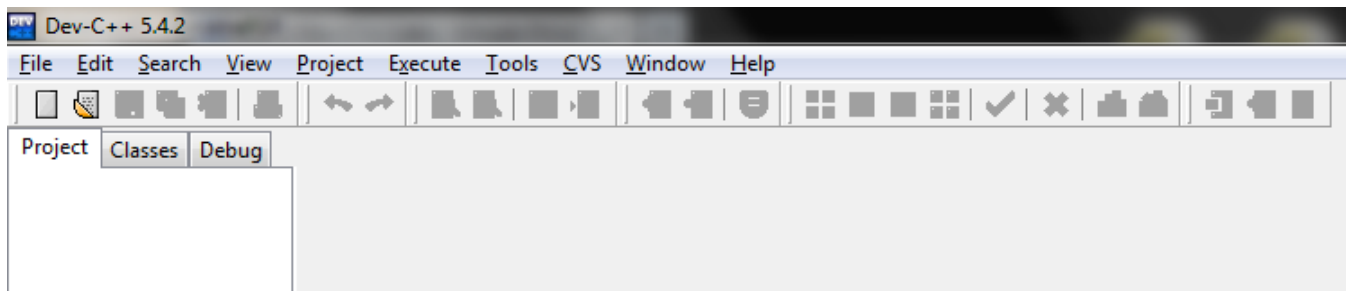
Στο εργαστήριο αυτό, θα ασχοληθούμε με δύο προγραμματιστικά περιβάλλοντα για τη γλώσσα C: τον gcc μεταγλωττιστή της C σε περιβάλλον Linux, και το περιβάλλον ανάπτυξης Dev-C++ για Windows. Επίσης, θα χρησιμοποιήσουμε και το πρόγραμμα WinSCP για να μεταφέρουμε αρχεία από υπολογιστές Windows στον λογαριασμό μας στην σχολή (και αντίστροφα). Τέλος, θα ασχοληθούμε με το πρώτο πρόγραμμα μας σε γλώσσα C, το οποίο θα μεταγλωττίσουμε, θα εκτελέσουμε και θα πειραματιστούμε με την έξοδό του.


### 1. Το περιβάλλον προγραμματισμού Dev-C++

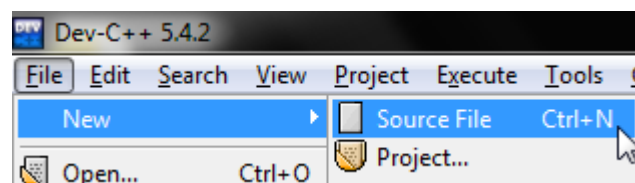
Στους λογαριασμούς των Windows συστημάτων της σχολής είναι διαθέσιμο το IDE Dev-C++, ένα γραφικό περιβάλλον ανάπτυξης κώδικα που απλοποιεί τη διαδικασία συγγραφής και μεταγλώττισης κώδικα (“αποκρύπτοντάς” μας την ύπαρξη και λειτουργία του gcc μεταγλωττιστή). Μπορούμε να το εγκαταστήσουμε στον προσωπικό μας υπολογιστή, κατεβάζοντας την τελευταία έκδοση από εδώ:

<http://sourceforge.net/projects/orwelldevcpp/files/latest/download>

Γράφουμε Dev-C++ και εκτελούμε στην έναρξη ή επιλέγουμε το σχετικό εικονίδιο από το μενού προγραμμάτων για να ανοίξουμε το περιβάλλον προγραμματισμού Dev-C++.



Για να δημιουργήσουμε ένα νέο αρχείο κώδικα, κάνουμε κλικ στο “File” -> “New” -> “Source File”, ή στο εικονίδιο .




Στο κέντρο της οθόνης δημιουργείται ένα κενό αρχείο με όνομα καρτέλας “Untitled1” στο οποίο μπορούμε να πληκτρολογήσουμε τον κώδικα του πρώτου προγράμματός μας!


Ας γράψουμε εδώ λοιπόν, το πρώτο μας πρόγραμμα C:

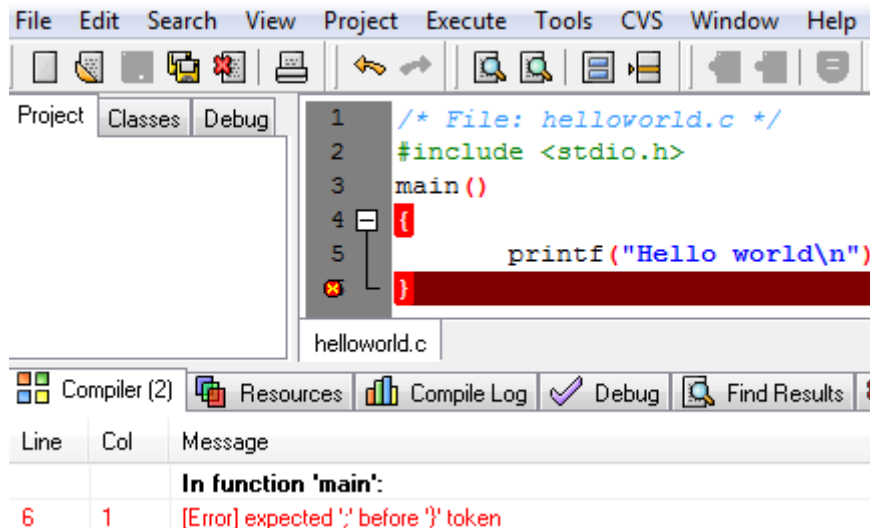
```
/* File: helloworld.c */  
  
#include <stdio.h>  
  
main()  
{  
    printf("Hello world\n");  
}
```

Αφού ολοκληρώσουμε την πληκτρολόγηση, πρέπει να αποθηκεύσουμε το αρχείο κώδικα στον δίσκο μας (το αστερί στα αριστερά της καρτέλας υποδηλώνει πως έχουν γίνει αλλαγές που δεν έχουν σωθεί).

```
[*] Untitled1
1  /* File: helloworld.c */
2  #include <stdio.h>
3  main()
4  {
5      printf("Hello world\n");
6  }
7
```

Για να το αποθηκεύσουμε, επιλέγουμε “File” -> “Save” ή το κουμπί , ώστε να ανοίξει το σχετικό παράθυρο διαλόγου. Εκεί, πρώτα πηγαίνουμε στη λίστα “Save as type” και επιλέγουμε “C source files (\*.c)”. Έπειτα, πλοηγούμαστε στη τοποθεσία (π.χ. Επιφάνεια Εργασίας) όπου μας ενδιαφέρει να αποθηκεύσουμε το αρχείο. Τέλος, δίνουμε ένα κατάλληλο όνομα στο αρχείο (π.χ. helloworld.c) και αποθηκεύουμε.

Για να μεταγλωττίσουμε το πρόγραμμα μας, επιλέγουμε “Execute” -> “Compile” ή πατάμε το F9, ή πατάμε το εικονίδιο . Αν το πρόγραμμα μας έχει συντακτικά λάθη ή επισημάνσεις, τότε στο κάτω μέρος της οθόνης θα εμφανίζονται σχετικά μηνύματα του μεταγλωττιστή, τα οποία περιγράφουν τη φύση του προβλήματος και μας καθοδηγούν για την επίλυσή του. Αφού εκτελέσουμε το μεταγλωττιστή, τότε θα εμφανιστούν μηνύματα όπως στην ακόλουθη εικόνα.



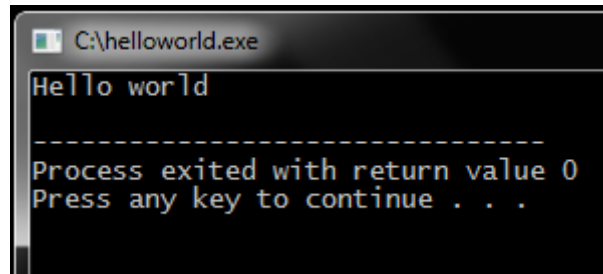
```
File Edit Search View Project Execute Tools CVS Window Help
Project Classes Debug
1  /* File: helloworld.c */
2  #include <stdio.h>
3  main()
4  {
5      printf("Hello world\n")
6  }
```

Line	Col	Message
6	1	In function 'main': [Error] expected ';' before '}' token

Το πλαίσιο μας πληροφορεί για την ύπαρξη συντακτικού λάθους στην γραμμή 6, πριν το δεξί άγκιστρο. Όντως, πριν από αυτό το άγκιστρο έχουμε ξεχάσει να πληκτρολογήσουμε το ερωτηματικό για να δηλώσουμε το τέλος της εντολής. Υπάρχει περίπτωση, όπως με παλαιότερη έκδοση του λογισμικού, να δούμε πιο γενικά μηνύματα λαθών, όπως απλά ‘Syntax error before “)” token’.

Αμέσως λοιπόν μετά από μία επιτυχή μεταγλώττιση, για να εκτελέσουμε το πρόγραμμά μας, επιλέγουμε “Execute” -> “Run”. Τότε, ανοίγει ένα παράθυρο του κελύφους εντολών των Windows (cmd.exe) στο οποίο περιέχονται οι εκτυπώσεις εκτέλεσης του προγράμματός μας. Αφού το πρόγραμμα ολοκληρώνεται με την εκτέλεση του μηνύματος, υπό κανονικές συνθήκες το παράθυρο

θα έκλεινε αυτόματα και άμεσα. Το Dev-C++ παρέχει τη δυνατότητα εισαγωγής παύσης μετά το τερματισμό του προγράμματος, μέχρι να πατηθεί οποιοδήποτε πλήκτρο.



```
C:\helloworld.exe
Hello world
-----
Process exited with return value 0
Press any key to continue . . .
```

Το Dev-C++ παρέχει πληθώρα διευκολύνσεων, όπως τη δημιουργία “project” για τη διαχείριση πολλαπλών αρχείων πηγαίου κώδικα, οπτική αποσφαλμάτωση του προγράμματός μας, προτάσεις αυτόματης συμπλήρωσης κλπ, τα οποία θα δούμε σε επόμενα εργαστήρια.

## 2. Η εφαρμογή WinSCP για μεταφορά αρχείων

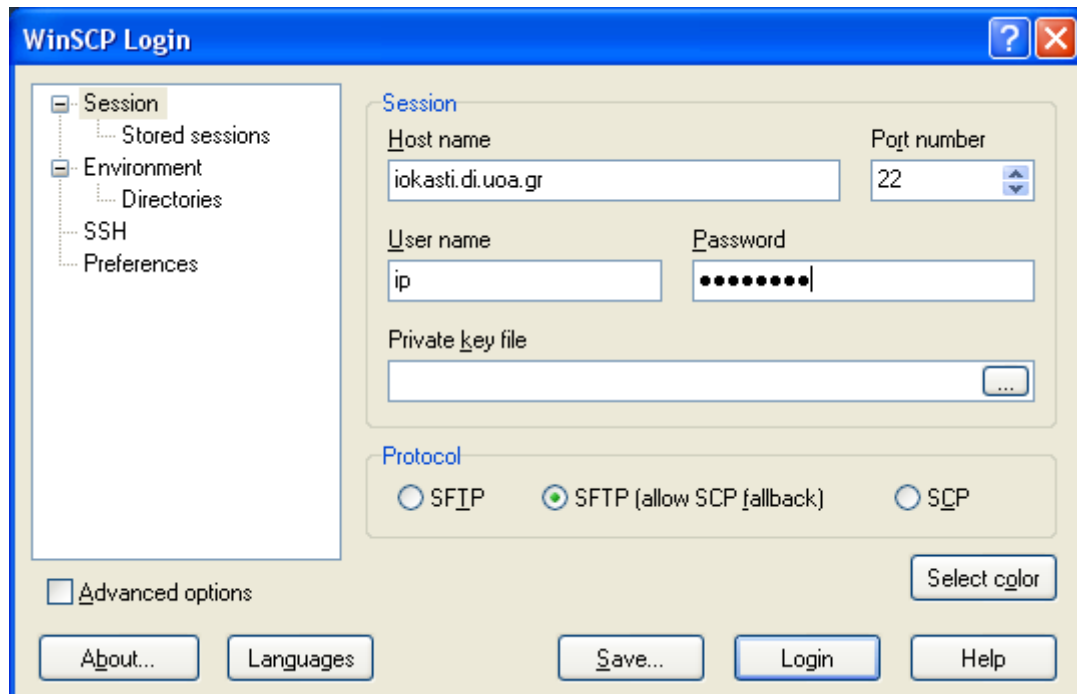
Θα χρησιμοποιήσουμε την εφαρμογή WinSCP για την μεταφορά αρχείων από υπολογιστές Windows στους υπολογιστές των εργαστηρίων UNIX της σχολής. Με τον τρόπο αυτό, μπορούμε να χρησιμοποιούμε τον υπολογιστή μας, ή τους υπολογιστές του εργαστηρίου Windows για να γράφουμε τα προγράμματά μας, να μεταφέρουμε τα αρχεία μας στο Unix και τελικά να ελέγχουμε την ορθή λειτουργία τους με χρήση του μεταγλωττιστή gcc, που είναι και η επίσημη πλατφόρμα εξέτασης του μαθήματος.

Για να εκτελέσουμε το WinSCP, κάνουμε στα εργαστήρια της σχολής Start->Run και πληκτρολογούμε WinSCP.

Από το σπίτι μας, μπορούμε να κατεβάσουμε το πρόγραμμα από την διεύθυνση:

<http://sourceforge.net/projects/winscp/files/latest/download>

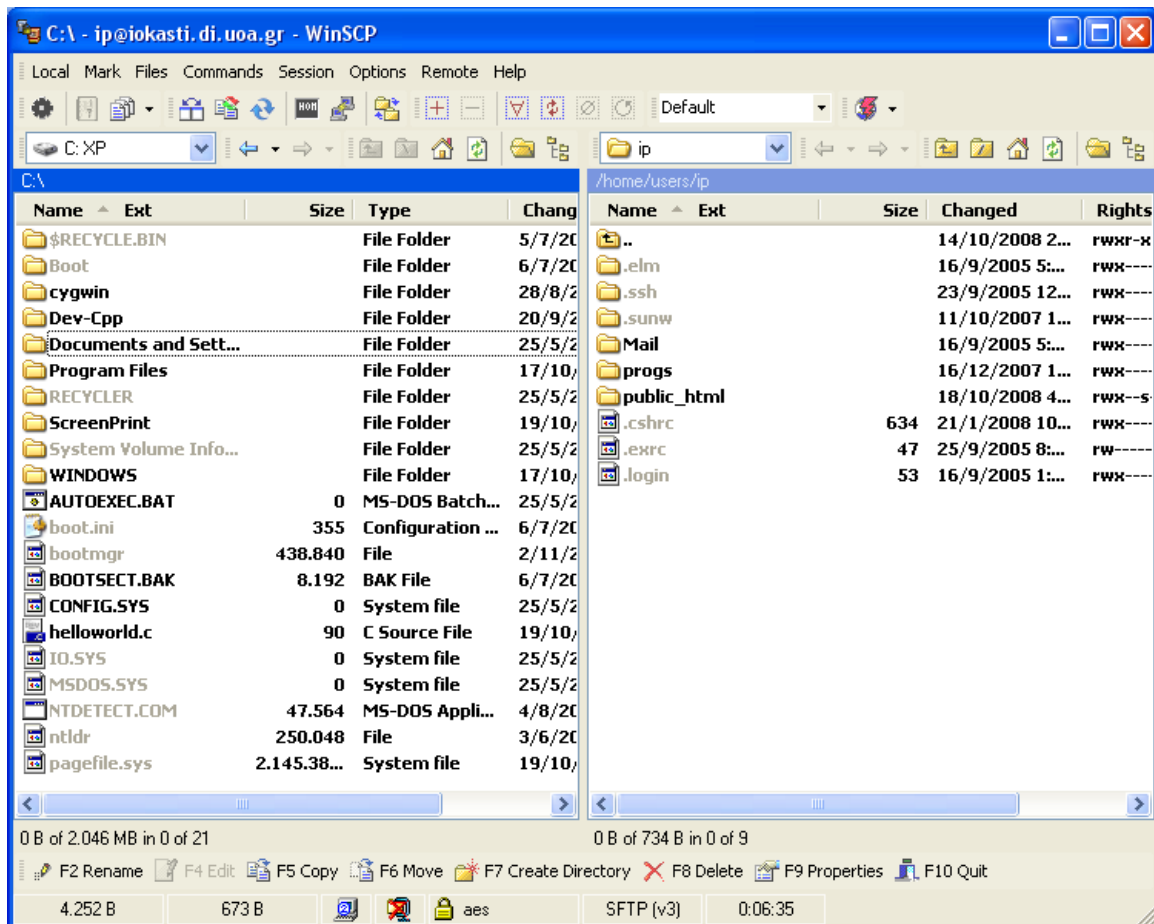
Εκτελώντας το πρόγραμμα βλέπουμε την ακόλουθη οθόνη:



όπου πληκτρολογούμε τα στοιχεία σύνδεσης μας δηλαδή:

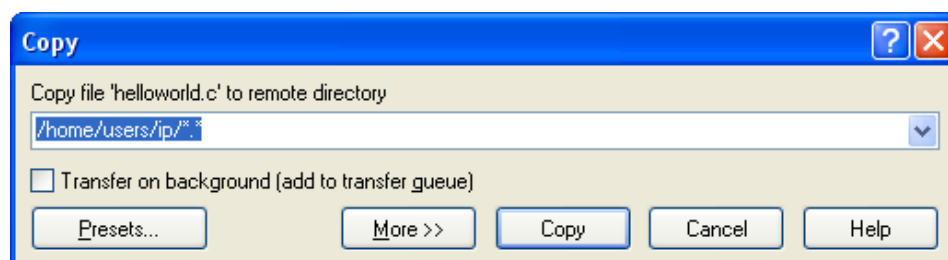
- Τον υπολογιστή που θα συνδεθούμε ([http://cgi.di.uoa.gr/~ip/linux\\_lab\\_machines.html](http://cgi.di.uoa.gr/~ip/linux_lab_machines.html))
- Το όνομα χρήστη
- Τον κωδικό μας

Και πατάμε το πλήκτρο “Login” οπότε και εμφανίζεται η ακόλουθη οθόνη:



Στο αριστερό μέρος της οθόνης φαίνονται τα περιεχόμενα του τοπικού καταλόγου μας και στο δεξί μέρος της οθόνης φαίνονται τα περιεχόμενα του λογαριασμού μας της σχολής.

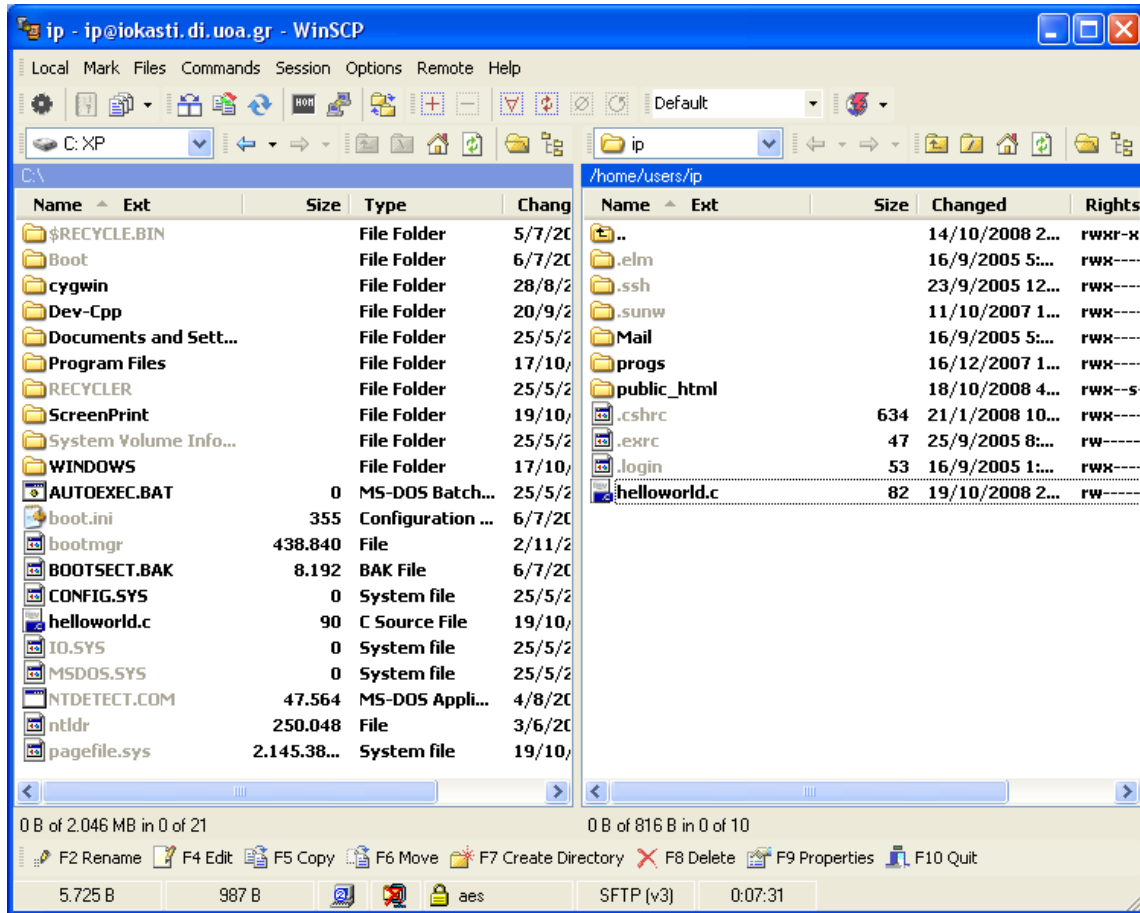
Έτσι, για να μεταφέρουμε αρχεία από τον υπολογιστή μας, στον λογαριασμό της σχολής, επιλέγουμε πρώτα τα αρχεία από το αριστερό μέρος της οθόνης και έπειτα πατάμε το πλήκτρο Copy ή πατάμε το πλήκτρο για συντόμευση F5. Εμφανίζεται τότε το ακόλουθο μήνυμα:



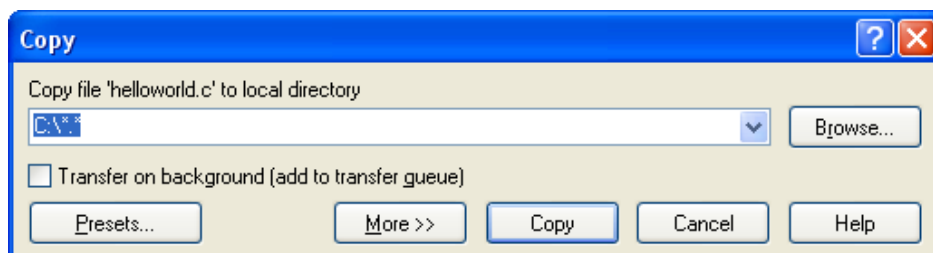
Με αυτό το μήνυμα ζητείται η επιβεβαίωση μας για την μεταφορά του αρχείου από τον τοπικό κατάλογο στον χώρο του λογαριασμού μας της σχολής. Αν πατήσουμε "Copy" το αρχείο μεταφέρεται στον λογαριασμό μας.

Βεβαίως είναι εφικτό να ακολουθήσουμε και την αντίστροφη διαδικασία, για να αντιγράψουμε αρχεία από τον λογαριασμό μας στην σχολή, στον τοπικό δίσκο.

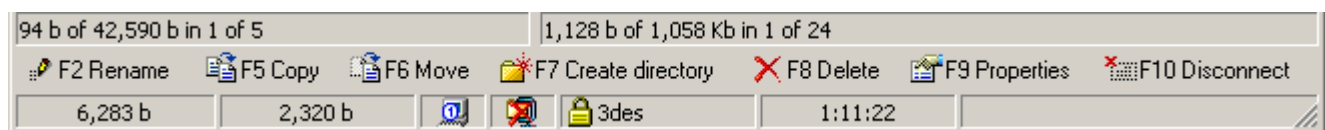
Για να το κάνουμε αυτό επιλέγουμε το αρχείο που μας ενδιαφέρει από το δεξί τμήμα της οθόνης και πατάμε το κουμπί «Copy».



Και πατάμε «Copy» στο επιβεβαιωτικό παράθυρο που εμφανίζεται:



Το WinSCP μας παρέχει και άλλες πρόσθετες δυνατότητες που φαίνονται στο κάτω μέρος της οθόνης:



όπως μετονομασία των αρχείων, δημιουργία καταλόγων, διαγραφή αρχείων και καταλόγων κ.λ.π.

Όταν ολοκληρώσουμε τις εργασίες μας, πατάμε το κουμπί «Disconnect» για να αποσυνδεθούμε.



### 3. Μεταγλώττιση προγραμμάτων σε περιβάλλον Unix

Κάνουμε login σε περιβάλλον Unix, όπου θα πρέπει να υπάρχει το αρχείο `helloworld.c` που μόλις μεταφέραμε χρησιμοποιώντας το πρόγραμμα WinSCP.

1. Μεταγλωττίστε το αρχείο `helloworld.c` με χρήση του `gcc`, ώστε να παραγάγετε το εκτελέσιμο αρχείο `helloworld`.

2. Εκτελέστε το πρόγραμμα `helloworld`.

Ας ρίξουμε τώρα μια πιο εκτενή ματιά στο πρόγραμμα `helloworld.c`.

```
/* File: helloworld.c */  
  
#include <stdio.h>  
  
main()  
{  
    printf("Hello world\n");  
}
```

Επεξηγήσεις:

Η συνάρτηση `printf()` εμφανίζει στην οθόνη την συμβολοσειρά που δέχεται σαν όρισμα.

Το `'\n'` είναι ο χαρακτήρας αλλαγής γραμμής.

4. Τροποποιείτε την συμβολοσειρά που δέχεται η `printf()` σαν όρισμα, ώστε η έξοδος που εμφανίζεται στην οθόνη να είναι:

```
Hello  
world
```

5. Χρησιμοποιείτε μία δεύτερη `printf()` για να έχετε την εξής έξοδο στο πρόγραμμά σας:

```
Hello  
world  
at  
d.i.t.
```

## ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 1<sup>η</sup>)

Όταν καλούμαστε να δημιουργήσουμε ένα πρόγραμμα για τη λύση ενός προβλήματος είναι σχεδόν απίθανο να είναι σωστό εξ αρχής. Οποιοσδήποτε, όσο έμπειρος κι αν είναι, θα έχει στο πρόγραμμά του λάθη, τα ονομαζόμενα bugs, τα οποία μπορεί να μην επιτρέπουν τη μεταγλώττιση του προγράμματος ή/και να το κάνουν να μη δουλεύει με τον επιθυμητό τρόπο. Στο σημερινό εργαστήριο θα εστιάσουμε στα συντακτικά λάθη, τα οποία αποτελούν μία μορφή τέτοιων σφαλμάτων, και θα δούμε χρήσιμες τεχνικές για την εύρεση και τη διόρθωσή τους.

### Συντακτικά λάθη

Ένα συντακτικό λάθος είναι, όπως υπαγορεύει και το όνομά του, ένα λάθος στη σύνταξη του προγράμματός μας. Για να καταλάβει ο μεταγλωττιστής τα προγράμματά μας πρέπει να είναι γραμμένα με έναν πολύ αυστηρό τρόπο. Οποιαδήποτε παράλειψη σε αυτόν τον αυστηρό τρόπο σύνταξης θα έχει ως αποτέλεσμα την αποτυχία της μεταγλώττισης.

Συχνά συντακτικά λάθη είναι η παράλειψη κάποιας παρένθεσης, το μη κλείσιμο κάποιας αγκύλης, η χρήση μιας μεταβλητής που δεν έχουμε δηλώσει, κλπ. Ας δούμε ένα παράδειγμα:

```
#include <stdio.h>

main() {
    printf("Hello world!\n");
}
```

Αν δώσουμε τον παραπάνω κώδικα προς μεταγλώττιση θα πάρουμε το εξής σφάλμα

```
4  missing terminating " character
5  syntax error before '}' token
```

Αυτό μας πληροφορεί ότι υπάρχει ένα σφάλμα στη γραμμή 4, το οποίο είναι ότι λείπει ένας χαρακτήρας `"`, καθώς και ότι υπάρχει ένα συντακτικό λάθος στη γραμμή 5 πριν το `}`.

Ας δούμε πρώτα το σφάλμα στη γραμμή 4. Με βάση αυτό, έχουμε παραλείψει ένα χαρακτήρα `"`. Όντως, αν το κοιτάξουμε καλά, λείπει το κλείσιμο του `"` στη συμβολοσειρά `Hello world!\n`. Φτιάχνοντας αυτό, αυτόματα φεύγει και το δεύτερο λάθος, το οποίο φανταστήκαμε ότι σχετίζεται με το πρώτο, αφού πριν το `}` στη γραμμή 5 είναι η γραμμή 4, στην οποία ήδη έχουμε υπόψη μας ένα σφάλμα.

Πειραματιστείτε με το παραπάνω πρόγραμμα για να δείτε τα διάφορα συντακτικά λάθη που μπορεί να δημιουργηθούν. Αφαιρέστε το τελικό ερωτηματικό στη συνάρτηση `printf`, αφαιρέστε το `f` απ' το `printf`, γράψτε λάθος το όνομα της `main`, ξεχάστε το `#` στο `include` και ό,τι άλλο σκεφτείτε. Δείτε τα μηνύματα που σας δίνει ο μεταγλωττιστής και προσπαθήστε να καταλάβετε πως σχετίζονται με αυτό που κάνατε. Ήταν όλα τα μηνύματα που σας έβγαλε κατατοπιστικά;

### Τεχνικές εύρεσης και διόρθωσης συντακτικών λαθών

Όπως είδαμε και στο προηγούμενο παράδειγμα, τα συντακτικά λάθη είναι εύκολο να τα ανακαλύψουμε αν διαβάσουμε τα μηνύματα του μεταγλωττιστή. Αυτή είναι και η βασική τακτική που χρησιμοποιούμε για να τα εντοπίσουμε και να τα διορθώσουμε. Κάθε μήνυμα του μεταγλωττιστή θα αναφέρει τη γραμμή όπου υπάρχει το συντακτικό λάθος καθώς και μια περιγραφή του. Από αυτά τα δύο στοιχεία μπορούμε, τις περισσότερες φορές, να βρούμε το συντακτικό λάθος.

Ας δούμε όμως, το επόμενο παράδειγμα:

```
#include <stdio.h>

main() {
    printf("Hello world!\n");
```

Αν δώσουμε τον παραπάνω κώδικα προς μεταγλώττιση θα πάρουμε το εξής σφάλμα

```
4 syntax error at end of input
```

Το οποίο μας λέει ότι υπάρχει ένα συντακτικό λάθος στη γραμμή 4 χωρίς καμία επιπλέον υπόδειξη. Το λάθος είναι ότι δεν έχουμε κλείσει την αγκύλη της `main`, οπότε ίσως να αναμέναμε κάτι σαν `"syntax error, bracket needed"`.

Οπότε, είναι εμφανές ότι το να βασιζόμαστε μόνο στα μηνύματα του μεταγλωττιστή για να διορθώσουμε τα συντακτικά λάθη δεν είναι μια τακτική που αποδίδει πάντα. Αυτό που χρειάζεται είναι εμπειρία ώστε να αποκτηθεί εξοικείωση με τα διάφορα συντακτικά λάθη, αλλά και προσοχή κατά τη συγγραφή του κώδικα ώστε να αποφεύγουμε επιπολαιότητες.



## ΕΡΓΑΣΤΗΡΙΟ 4: Μεταβλητές, Δομές Ελέγχου και Επανάληψης

Στο εργαστήριο αυτό, θα εξοικειωθούμε με τους τύπους δεδομένων που μας παρέχει η γλώσσα C, θα χρησιμοποιήσουμε τις δομές επανάληψης (for, while, do...while), την δομή ελέγχου (if...else) και θα μάθουμε πώς να διαχειριζόμαστε την έξοδο του προγράμματος μας, μέσω της συνάρτησης printf().

### Άσκηση 1: Υπολογισμός σειράς

1.1. Γράψτε πρόγραμμα C που να υπολογίζει το άθροισμα της σειράς

$$\sum_{i=1}^{100} i$$

και να το εκτυπώνει στην οθόνη. Να χρησιμοποιηθεί η δομή επανάληψης do...while.

Δομές επανάληψης while και do...while:

WHILE	DO...WHILE
<pre>while (E<sub>2</sub>) { ..... }</pre>	<pre>do { ..... } while (E<sub>2</sub>);</pre>
E <sub>2</sub> : Συνθήκη Εισόδου και Συνέχισης Επανάληψης	E <sub>2</sub> : Συνθήκη Συνέχισης Επανάληψης

Η δομή επανάληψης for:

FOR	Όπου
<pre>for (E<sub>1</sub>; E<sub>2</sub>; E<sub>3</sub>) { ..... }</pre>	E <sub>1</sub> : Εντολή Αρχικοποίησης E <sub>2</sub> : Συνθήκη Εισόδου και Συνέχισης Επανάληψης (εφόσον ισχύει, εισερχόμαστε στον βρόχο ή επαναλαμβάνεται ο βρόχος) E <sub>3</sub> : Εντολή Επανάληψης (εκτελείται στο τέλος της κάθε επανάληψης και πριν αρχίσει η επόμενη)

Ισοδύναμη δομή while με την δομή for:

FOR	WHILE
<pre>for (E<sub>1</sub>; E<sub>2</sub>; E<sub>3</sub>) { ..... }</pre>	<pre>E<sub>1</sub>; while (E<sub>2</sub>) { ..... E<sub>3</sub>; }</pre>

## Τύποι Δεδομένων

`int` (ακέραιος, συνήθως με 4 bytes)`long` (ακέραιος, τουλάχιστον με 4 bytes)

(βλέπε και σημειώσεις μαθήματος, σελ. 31)

Η συνάρτηση `printf()` μπορεί να εκτυπώσει την τιμή μεταβλητών που δέχεται σαν όρισμα.

Σύνταξη: `printf("%y", var);`όπου `y` είναι σύμβολο που αντιστοιχεί στον τύπο δεδομένων της μεταβλητής `var`.

<code>d: int</code>	<code>f: float, double</code>
<code>ld: long</code>	

**1.2.** Τροποποιήστε το πρόγραμμά σας, ώστε να υπολογίζει το άθροισμα:

$$\sum_{i=1}^{100} \frac{1}{i^2}$$

και να το εκτυπώνει στην οθόνη.

Υπόδειξη: Χρησιμοποιήστε μία ενδιάμεση μεταβλητή για να αποθηκεύετε προσωρινά τον τρέχοντα όρο της σειράς.

## Τύποι Δεδομένων

`float` (πραγματικός αριθμός, συνήθως με 4 bytes)`double` (πραγματικός αριθμός, συνήθως με 8 bytes)

**1.3.** Αν γνωρίζετε ότι

$$\pi = \sqrt{6 \sum_{i=1}^{+\infty} \frac{1}{i^2}}$$

τροποποιήστε το πρόγραμμα σας, ώστε να υπολογίσετε το  $\pi$  μέσω των πρώτων 100 όρων της σειράς.

`double sqrt(double)`: Επιστρέφει την τετραγωνική ρίζα του αριθμού που δέχεται σαν όρισμα.

Απαιτείται η ενσωμάτωση του αρχείου επικεφαλίδας `math.h`.

Στην μεταγλώττιση με τον `gcc` απαιτείται και η επιλογή `-lm`.

**1.4.** Παρατηρήστε ότι οι όροι που προστίθενται στην σειρά ολοένα και γίνονται πιο μικροί, με αποτέλεσμα από ένα σημείο και μετά να είναι αρκούντως μικροί για να μην επηρεάζουν το αποτέλεσμα, αφού χρησιμοποιούμε αριθμητική πεπερασμένης ακρίβειας. Αλλάξτε το κριτήριο τερματισμού του υπολογισμού, ώστε ο υπολογισμός να σταματάει όταν ο τρέχων όρος που προστίθεται στην σειρά είναι μικρότερος από  $10^{-15}$ .

**1.5.** Τροποποιήστε το πρόγραμμά σας, ώστε να εκτυπώνεται στην οθόνη το αποτέλεσμα με ακρίβεια 8 δεκαδικών ψηφίων.

Η συνάρτηση `printf()` μπορεί να εκτυπώσει την τιμή πραγματικών μεταβλητών που δέχεται σαν όρισμα με επιθυμητή μορφοποίηση.

Σύνταξη: `printf("%a.bf", var);`

a: Ορίζουμε το πλήθος των θέσεων που θα γίνει η εκτύπωση.

b: Ορίζουμε το πλήθος των ψηφίων μετά την υποδιαστολή που θα εκτυπωθούν.

**1.6** Τροποποιήστε το πρόγραμμα σας, ώστε να υπολογίζει την σειρά:

$$S_1 = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \frac{1}{6^2} + \dots$$

**1.7** Επαναλάβετε και για τις σειρές που εμφανίζονται στις σημειώσεις του μαθήματος, σελ. 24.

**Άσκηση 2:** Υπολογισμός ριζών τριωνύμου

**2.1** Γράψτε πρόγραμμα C που να παράγει τρεις τυχαίους πραγματικούς αριθμούς στο διάστημα  $[0,1]$  και να τους αποθηκεύει στις μεταβλητές τύπου `double` `a`, `b` και `c`, τις οποίες στη συνέχεια να εκτυπώνει.

<pre>int rand()</pre> <p>Απαιτείται ενσωμάτωση του αρχείου επικεφαλίδας <code>stdlib.h</code></p> <p>Επιστρέφει έναν ψευδο-τυχαίο ακέραιο από 0 έως <code>RAND_MAX</code> (συμβολική σταθερά που ορίζεται στο αρχείο επικεφαλίδας <code>stdlib.h</code>)</p>	<pre>srand(unsigned int seed)</pre> <p>Απαιτείται ενσωμάτωση του αρχείου επικεφαλίδας <code>stdlib.h</code></p> <p>Αρχικοποιεί την γεννήτρια ψευδο-τυχαίων αριθμών με βάση τον αριθμό/φύτρα 'seed'.</p> <p>Συνήθως χρησιμοποιούμε σα φύτρα την τρέχουσα ώρα, όπως επιστρέφεται από την κλήση της συνάρτησης <code>time(NULL)</code> (απαιτείται ενσωμάτωση του αρχείου επικεφαλίδας <code>time.h</code>)</p>
--	--

**2.2** Γράψτε πρόγραμμα C που να υπολογίζει τις πραγματικές ρίζες ενός τριωνύμου. Τα δεδομένα εισόδου (συντελεστές του τριωνύμου) θα είναι τυχαίες πραγματικές μεταβλητές `a`, `b`, `c` στο διάστημα  $[0,1]$ . Αν το τριώνυμο δεν έχει πραγματικές ρίζες, να εκτυπώνεται ένα σχετικό μήνυμα ενημέρωσης. Οι ρίζες να εκτυπώνονται με ακρίβεια 3 δεκαδικών ψηφίων.

<p>Η δομή ελέγχου <code>if ... else</code></p> <pre> if (C)     E<sub>1</sub> else     E<sub>2</sub> </pre> <p>Εάν η συνθήκη <code>C</code> είναι αληθής, τότε εκτελείται η εντολή (ή μπλοκ εντολών) <code>E<sub>1</sub></code>, αλλιώς η εντολή (ή block εντολών) <code>E<sub>2</sub></code></p>
---

**2.3** Ενσωματώστε στο πρόγραμμά σας και την περίπτωση το τριώνυμο να έχει μιγαδικές ρίζες, οπότε θα πρέπει να τις υπολογίζει.



**Άσκηση 3:** Υπολογισμός ημέρας μίας δεδομένης ημερομηνίας

Ο ακόλουθος αλγόριθμος υπολογίζει την ημέρα που αντιστοιχεί σε μία δεδομένη ημερομηνία:

Έστω η ημερομηνία  $DD/MM/YYYY$

Αν  $MM \leq 2$  τότε

$$NYYYY = YYYY - 1$$

$$NMM = 0$$

Αν  $MM > 2$  τότε

$$NYYYY = YYYY$$

$$NMM = \left\lfloor \frac{4 * MM + 23}{10} \right\rfloor$$

$$IDAY = 365 * YYYY + DD + 31 * (MM - 1) - NMM + \left\lfloor \frac{NYYYY}{4} \right\rfloor - \left\lfloor \frac{3}{4} * \left( \left\lfloor \frac{NYYYY}{100} \right\rfloor + 1 \right) \right\rfloor$$

Αν  $IDAY \bmod 7 = 0$  τότε  $DAY = Saturday$

Αν  $IDAY \bmod 7 = 1$  τότε  $DAY = Sunday$

.....

Αν  $IDAY \bmod 7 = 6$  τότε  $DAY = Friday$

Με  $\lfloor x \rfloor$  συμβολίζουμε τον μέγιστο ακέραιο αριθμό που δεν είναι μεγαλύτερος του αριθμού  $x$ .

Γράψτε πρόγραμμα C που να βρίσκει τι ημέρα γεννηθήκατε (ενσωματώστε την ημερομηνία γέννησής σας μέσα στο πρόγραμμα).

## ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 2<sup>η</sup>)

Στο προηγούμενο εργαστήριο είδαμε την μέθοδο για να ανιχνεύουμε και να διορθώνουμε συντακτικά λάθη. Στο παρόν εργαστήριο θα συνεχίσουμε τη συζήτηση για την αποσφαλμάτωση προγραμμάτων, εστιάζοντας αυτή τη φορά στα λάθη λογικής.

### Λογικά λάθη

Ένα λογικό λάθος είναι ένα λάθος που, ενώ επιτρέπει την επιτυχή μεταγλώττιση και εκτέλεση του προγράμματος, τελικά κάνει τα αποτελέσματά του να μην είναι σωστά. Αυτό μπορεί να οφείλεται είτε σε κάποιο λάθος στην αρχική μας σκέψη για το πως θα λύσουμε το πρόβλημα, είτε ακόμα και στη λανθασμένη υλοποίηση μιας σωστής σκέψης.

Για παράδειγμα ας θεωρήσουμε το πρόβλημα να υπολογίσουμε το άθροισμα  $1+3+5+\dots$  για τους πρώτους 20 όρους. Ένα πρόγραμμα που ευελπιστεί να κάνει αυτή τη δουλειά είναι το εξής

```
#include <stdio.h>
#define N 20

main() {
    int S=0, a=1;
    while (a<=N) {
        S = S+a;
        a = a+2;
    }
    printf("%d\n", S);
}
```

Το προηγούμενο πρόγραμμα μεταγλωττίζεται με απόλυτη επιτυχία και εμφανίζει αποτελέσματα. Το πρόβλημα είναι ότι το αποτέλεσμα που εμφανίζει, το 100, δεν είναι σωστό. Εμείς θα αναμέναμε το 400. Άρα στο πρόγραμμά μας υπάρχει κάποιο λογικό λάθος το οποίο πρέπει να αναζητήσουμε και να διορθώσουμε.

### Τεχνικές εύρεσης και διόρθωσης λογικών λαθών

Τα λογικά λάθη είναι, κατά γενική ομολογία, πολύ δύσκολο να εντοπιστούν. Ενώ με τα συντακτικά λάθη μπορούμε να αντλήσουμε πολύτιμες πληροφορίες από τα μηνύματα του μεταγλωττιστή, με τα λογικά λάθη όλα θα λειτουργούν σωστά, με εξαίρεση ότι τελικά θα παίρνουμε λανθασμένα αποτελέσματα (ή και καθόλου αποτελέσματα).

Όταν αντιμετωπίζουμε ένα λογικό λάθος το πρώτο που πρέπει να κάνουμε είναι να προσπαθήσουμε να εντοπίσουμε τις πιθανές πηγές του. Για να το πετύχουμε αυτό πρέπει να δούμε με προσοχή τα αποτελέσματα που εκτυπώνει το πρόγραμμά μας και να σκεφτούμε πιθανές αιτίες που μπορεί να οδήγησαν σε τέτοιου είδους αποτελέσματα. Για παράδειγμα, αν θέλουμε να εκτυπώνουμε κάποια στοιχεία με βάση ένα κριτήριο και καταλήγουμε να εκτυπώνουμε όλα τα στοιχεία, τότε πιθανότατα το πρόβλημα βρίσκεται στον τρόπο που έχουμε υλοποιήσει το κριτήριο επιλογής.

Έχοντας σκεφτεί περίπου τι μπορεί να φταίει εντοπίζουμε το επίμαχο κομμάτι του κώδικα και το ξανακοιτάμε με προσοχή. Μήπως κάτι που έχουμε γράψει δε συνάδει με την αρχική μας σκέψη; Μήπως έχουμε παραλείψει κάτι;

Αν είναι δύσκολο να δούμε με το μάτι τι μπορεί να φταίει τότε πρέπει να καταφύγουμε σε μια "ιχνηλάτηση" του προγράμματος. Στο επίμαχο κομμάτι του κώδικα και για όσες μεταβλητές περιλαμβάνει εκτυπώνουμε την τιμή τους στα διάφορα στάδια εκτέλεσης με τη χρήση συναρτήσεων

`printf`. Έτσι, μπορούμε να δούμε τις τιμές που παίρνουν και να αποκτήσουμε μια καλύτερη εικόνα για το στάδιο στο οποίο η υλοποίησή μας αρχίζει να χωλαίνει.

Ας επανέλθουμε στο προηγούμενο παράδειγμα και ας δούμε πως θα το αντιμετωπίζαμε με βάση όσα είπαμε. Αρχικά παρατηρούμε ότι το αποτέλεσμα που παίρνουμε δεν είναι σωστό, συγκεκριμένα είναι σημαντικά μικρότερο από αυτό που θα περιμέναμε. Αυτό μπορεί να οφείλεται είτε στο ότι δεν κάνουμε σωστά το άθροισμα, είτε στο ότι δεν παράγουμε σωστά τους όρους που αθροίζουμε, είτε στο ότι αθροίζουμε λιγότερους όρους απ' ό,τι θα θέλαμε.

Σε κάθε περίπτωση, όλα τα προηγούμενα συγκλίνουν στο ότι δεν κάνουμε κάτι σωστά μέσα στη δομή επανάληψης. Οι επίμαχες μεταβλητές είναι το `s` και το `a`, οπότε ας εμφανίζουμε τις τιμές τους με σωστή τοποθέτηση συναρτήσεων `printf` ως εξής

```
main() {
    int S=0, a=1;
    while (a<=N) {
        S = S+a;
        printf("%d %d\n", S, a);
        a = a+2;
    }
    printf("%d\n",S);
}
```

Εκτελώντας αυτόν τον κώδικα παίρνουμε σαν αποτέλεσμα

```
1 1
4 3
9 5
16 7
25 9
36 11
49 13
64 15
81 17
100 19
100
```

Από το αποτέλεσμα καταλαβαίνουμε ότι ο υπολογισμός του αθροίσματος, αλλά και των όρων, γίνεται σωστά. Άρα το πρόβλημα πρέπει να βρίσκεται στο πλήθος των όρων που χρησιμοποιούμε, που όπως βλέπουμε δεν είναι 20 όπως θα θέλαμε, αλλά 10. Οπότε αυτό που φταίει είναι το πλήθος των φορών που εκτελείται η δομή επανάληψης. Όμως αυτό εξαρτάται από τη συνθήκη της, άρα το λάθος πρέπει να βρίσκεται σε αυτή. Αν τη δούμε καλύτερα, λέει `while (a<=N)`, το οποίο σημαίνει ο τρέχων όρος να είναι μικρότερος του πλήθους των όρων που θέλουμε να αθροίσουμε, ενώ εμείς θέλουμε απλά να αθροίσουμε 20 όρους. Αυτό μπορούμε να το πετύχουμε αν αντικαταστήσουμε τη `while` με μια `for` της μορφής `for (i=1 ; i<=N ; i++)`.

Φυσικά, θα υπάρξουν και φορές όπου ο κώδικάς μας θα είναι απόλυτα σύμφωνος με όσα έχουμε σκεφτεί και φαινομενικά σωστός, όμως σε κάποια παραδείγματα δεν θα λειτουργεί σωστά. Τότε αυτό που έχει συμβεί είναι ότι δεν έχουμε καλύψει όλες τις πιθανές περιπτώσεις, μια υποχρέωση που είναι εκ των ων ουκ άνευ για έναν σωστό προγραμματιστή. Γι' αυτό πρέπει πάντα να δοκιμάζουμε εξονυχιστικά τα προγράμματά μας με διάφορα παραδείγματα, όχι μόνο απλά, αλλά και σύνθετα και ασυνήθιστα, για να βεβαιωθούμε ότι είναι όντως σωστά.

Αν κάποιος από τα παραδείγματα που δοκιμάσαμε δεν δίνει το αναμενόμενο αποτέλεσμα τότε έχουμε παραλείψει στον κώδικά μας να καλύψουμε μια τέτοια περίπτωση. Οπότε πρέπει να σκεφτούμε ποια είναι τα χαρακτηριστικά του συγκεκριμένου παραδείγματος που το κάνουν να μην εμπίπτει σε όσα έχουμε σκεφτεί και υλοποιήσει. Για να το πετύχουμε αυτό, είναι πολύ πιθανό ότι θα χρειαστούμε και πάλι να καταφύγουμε σε όσες τεχνικές περιγράψαμε πριν και να τοποθετήσουμε κατάλληλες `printf` σε κρίσιμα σημεία του προγράμματός μας. Μόλις βρούμε τι φταίει, εμπλουτίζουμε την αρχική μας σκέψη ώστε να καλύψει και αυτό το είδος περιπτώσεων και κάνουμε τις απαραίτητες προσθήκες στον κώδικά μας.

Γενικά, η διόρθωση λογικών λαθών απαιτεί εμπειρία, υπομονή και εξοικείωση με το πρόβλημα και τον κώδικά μας. Αν και περιγράψαμε τεχνικές για την εύρεση και διόρθωση λογικών σφαλμάτων, πρέπει να έχουμε κατά νου ότι κάθε λογικό λάθος είναι διαφορετικό από οποιοδήποτε άλλο. Ο τρόπος με τον οποίο θα φτάσουμε από τα λανθασμένα αποτελέσματα στην αιτία τους μέσα στον κώδικά μας απαιτεί αναλυτική σκέψη και δε γίνεται να υπάρξουν καθολικά εφαρμόσιμες τεχνικές γι' αυτό το σκοπό.

Σημείωση: Αν και αναφέραμε έναν τρόπο αποσφαλμάτωσης μέσω `printf`, αυτός ο τρόπος δεν είναι πάντα αποτελεσματικός ή εύκολα υλοποιήσιμος, ειδικά σε μεγάλα προγράμματα. Γι' αυτό το σκοπό υπάρχουν εξειδικευμένα εργαλεία, γνωστά ως `debuggers`, που βοηθούν πολύ τη διαδικασία αποσφαλμάτωσης. Είναι σημαντικό με την πρώτη ευκαιρία να εξοικειωθείτε με τη χρήση κάποιου τέτοιου προγράμματος, π.χ. με τον `gdb` που βρίσκεται εγκατεστημένος στα μηχανήματα Linux του Τμήματος. Παρουσίαση του `gdb` καθώς και του `debugger` του Dev-C++ θα γίνει στο Εργαστήριο 8.

## Άσκηση

Το παρακάτω πρόγραμμα ευελπιστεί να υπολογίζει την παράσταση  $1 - 1/2 + 1/4 - 1/8 + 1/16 - \dots$  για όσους όρους είναι μεγαλύτεροι από το `LIMIT`. Για να το πετύχει αυτό ο προγραμματιστής του έκανε την εξής σκέψη “Σε μια μεταβλητή `c` θα κρατάω τον τρέχοντα όρο που θα προστεθεί, ο οποίος παράγεται από τον προηγούμενο αν πολλαπλασιάσω με  $1/2$ . Θέλω ανά πάσα στιγμή ο όρος `c` να μην ξεπερνά το `LIMIT`. Το `c` έχει διαφορετικό πρόσημο σε κάθε όρο, οπότε θα χρησιμοποιήσω μια μεταβλητή `sign` που θα την πολλαπλασιάζω με το `-1` ώστε ανά δύο όρους να έχει το ίδιο πρόσημο. Τελικά θα εκτυπώσω το άθροισμα `s`”. Το πρόγραμμα αυτό όμως, έχει συντακτικά και λογικά λάθη. Μπορείτε να το αποσφαλματώσετε;

```
#include <stdio.h>
#define LIMIT 0.008

main() {
    float S=0, c=1;
    int sign=1;

    while (c<LIMIT) {
        S = S+c;
        sign = (-1)*sign;
        c = sign*1/2*c;
    }
    print("%f\n", s);
}
```

## ΕΡΓΑΣΤΗΡΙΟ 5: Είσοδος/Εξοδος Χαρακτήρων

Στο εργαστήριο αυτό, θα ασχοληθούμε με θέματα εισόδου/εξόδου και θα εξοικειωθούμε με τη χρήση συναρτήσεων που μας παρέχει η C για την είσοδο και την έξοδο χαρακτήρων (`getchar` και `putchar`).

**Άσκηση 1:** Κατασκευάστε πρόγραμμα C που να εκτυπώνει τους χαρακτήρες με ASCII κωδικό που είναι πολλαπλάσιο του 3, και μεταξύ 33 και 105, μαζί με τους κωδικούς αυτούς. Παρατηρήστε την έξοδο του προγράμματος σε σχέση με τον παρακάτω πίνακα των ASCII κωδικών:

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	\$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c		124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

### Άσκηση 2:

**2.1** Κατασκευάστε πρόγραμμα C που να εκτυπώνει στην οθόνη σχήμα με την ακόλουθη μορφή:

```
*  
**  
***  
****  
*****  
*****
```

Το πλήθος των γραμμών που θα εκτυπωθούν να το ορίσετε με `#define`.

Η συνάρτηση `putchar(int c)` εκτυπώνει τον χαρακτήρα που αντιστοιχεί στον ASCII κωδικό που δέχεται σαν όρισμα.

Ισοδύναμες χρήσεις: `putchar('*')` ή `putchar(42)`  
όπου 42 είναι ο ASCII κωδικός του χαρακτήρα `*`.

**2.2** Κατασκευάστε πρόγραμμα C που να εκτυπώνει στην οθόνη σχήμα με την ακόλουθη μορφή (και εδώ να χρησιμοποιήσετε `#define` για τον ορισμό του πλήθους των γραμμών που θα εκτυπωθούν):

```
*
***
*****
*****
*****
*****
*****
```

### Άσκηση 3:

**3.1** Αντιγράψτε το αρχείο `capitalize.c` από την ιστοσελίδα του μαθήματος, μεταγλωττίστε το και εκτελέστε το με είσοδο το ίδιο το πηγαίο αρχείο `capitalize.c`. Παρατηρήστε τη λειτουργία του.

```
/* File: capitalize.c */
#include <stdio.h>

main()
{ int ch; /* Be careful! Declare ch as int because of getchar() and EOF */
  ch = getchar(); /* Read first character */
  while (ch != EOF) { /* Go on if we didn't reach end of file */
    if (ch >= 'a' && ch <= 'z') /* If lower case letter */
      ch = ch - ('a'-'A'); /* Move 'a'-'A' positions in the ASCII table */
    putchar(ch); /* Print out character */
    ch = getchar(); /* Read next character */
  }
}
```

Η συνάρτηση `getchar()` διαβάζει έναν χαρακτήρα από την είσοδο και επιστρέφει τον ASCII κωδικό του χαρακτήρα. Αν δεν υπάρχει άλλος χαρακτήρας για διάβασμα στην είσοδο, επιστρέφει την ειδική ακέραια τιμή `EOF` που είναι ορισμένη στο `stdio.h`.

**3.2** Κατασκευάστε το πρόγραμμα `lowercase.c`, ώστε να κάνει το αντίστροφο, δηλαδή να μετατρέπει τους χαρακτήρες που εμφανίζονται με κεφαλαία γράμματα σε χαρακτήρες με μικρά γράμματα.

**3.3** Τροποποιήστε το πρόγραμμά σας, ώστε να μετατρέπονται ταυτόχρονα και οι κεφαλαίοι χαρακτήρες σε μικρούς και οι μικροί σε κεφαλαίους.

**3.4** Κωδικοποίηση κειμένου: Θα τροποποιήσουμε λίγο το πρόγραμμα μας και θα δημιουργήσουμε έναν απλοϊκό τρόπο για να κωδικοποιήσουμε ένα κείμενο.

**3.4.1** Δημιουργήστε ένα αρχείο κειμένου με όνομα `text` και πληκτρολογήστε σε αυτό ένα τυχαίο κείμενο.

**3.4.2** Τροποποιήστε το πηγαίο αρχείο `capitalize.c`, ώστε να διαβάζει από την είσοδο χαρακτήρες και, για καθένα απ' αυτούς, να εκτυπώνει στην έξοδο τον χαρακτήρα που έχει ASCII κωδικό αυξημένο κατά ένα σε σχέση με αυτόν του χαρακτήρα που διάβασε. Ονομάστε το νέο αρχείο `encode.c` και μεταγλωττίστε το για να δημιουργηθεί το εκτελέσιμο `encode`.

**3.4.3** Εκτελέστε το πρόγραμμα `encode` με ανακατεύθυνση εισόδου από το αρχείο `text`.

**3.4.4** Εκτελέστε το πρόγραμμα `encode` με ανακατεύθυνση εισόδου από το αρχείο `text` και ανακατεύθυνση εξόδου στο αρχείο `encoded_text`.

**3.4.5** Δημιουργήστε το πηγαίο αρχείο `decode.c` ώστε να διαβάζει από την είσοδο χαρακτήρες και, για καθένα απ' αυτούς, να εκτυπώνει τον χαρακτήρα με ASCII κωδικό μειωμένο κατά ένα σε σχέση με αυτόν που διάβασε.

**3.4.6** Μεταγλωττίστε το και εκτελέστε το με ανακατεύθυνση της εισόδου από το αρχείο `coded_text`.

**3.4.7** Εκτελέστε το πρόγραμμα `decode` για την αποκωδικοποίηση ενός κωδικοποιημένου κειμένου, έτσι ώστε να παίρνει την είσοδό του απ' ευθείας από το πρόγραμμα `encode`, το οποίο να καλείται έτσι ώστε να κωδικοποιεί το αρχείο `text`, χωρίς να χρησιμοποιήσετε το ενδιάμεσο αρχείο `coded_text`.





## ΕΡΓΑΣΤΗΡΙΟ 6: Συναρτήσεις και Αναδρομή

Στο εργαστήριο αυτό θα μάθουμε για τη χρήση συναρτήσεων με σκοπό την κατασκευή αυτόνομων τμημάτων προγραμμάτων που υλοποιούν μία συγκεκριμένη διαδικασία, τα οποία έχουν σαν στόχο τη δόμηση του προγράμματος σε ανεξάρτητα μέρη και, επιπροσθέτως, αποτελούν δομικούς λίθους που μπορούν να επαναχρησιμοποιηθούν και σε άλλα προγράμματα. Θα επεκτείνουμε τη συζήτηση μιλώντας για αναδρομικές συναρτήσεις, δηλαδή για συναρτήσεις που στο σώμα τους καλούν τον εαυτό τους. Επίσης, θα αναφερθούμε στις εξωτερικές ή καθολικές (global) μεταβλητές και θα δούμε πότε μπορούν να είναι χρήσιμες στον προγραμματισμό. Τέλος, σ' ένα παράρτημα, θα περιγράψουμε τη διαδικασία δημιουργίας projects στο Dev-C++, που είναι χρήσιμη όταν κάποιος θέλει να δουλεύει σ' αυτό το περιβάλλον και να έχει οργανωμένο ένα πρόγραμμα C σε πολλά πηγαία αρχεία και αρχεία επικεφαλίδας.

### Άσκηση 1: Το πρόβλημα $3n+1$

**1.1** Κατασκευάστε τη συνάρτηση `int isodd(int n)` που δέχεται σαν όρισμα έναν ακέραιο και επιστρέφει 1, αν ο αριθμός είναι περιττός, ή 0, αν ο αριθμός είναι άρτιος.

**1.2** Γράψτε σε γλώσσα C τον ακόλουθο (διατυπωμένο σε ψευδογλώσσα) αλγόριθμο, που επιδεικνύει ένα άλυτο μέχρι στιγμής μαθηματικό πρόβλημα, το λεγόμενο πρόβλημα του  $3n+1$ .<sup>1</sup>

Έστω τυχαίος ακέραιος  $n$   
Επανάλαβε όσο το  $n \neq 1$   
    Αν ο  $n$  είναι περιττός θέσε  $n = 3n+1$ , αλλιώς θέσε  $n = n/2$   
    Τύπωσε το  $n$

Ορισμός Συνάρτησης (δείτε και σημειώσεις μαθήματος, σελ. 59-61):

```
<Τύπος Επιστροφής> <Όνομα Συνάρτησης> (<Τυπικές Παράμετροι>)  
{  
    <Εντολές και Δηλώσεις>  
}
```

<Τύπος Επιστροφής>

Τύπος δεδομένων της τιμής που επιστρέφεται από τη συνάρτηση μέσω της εντολής:

```
return <παράσταση>;
```

Σε περίπτωση μη επιστροφής τιμής, ο <Τύπος Επιστροφής> ορίζεται σαν `void`.

<Τυπικές Παράμετροι>

Μεταβλητές, μαζί με τους τύπους τους, που χρησιμοποιούνται στη συνάρτηση, χωρισμένες με κόμμα, στις οποίες δίνονται τιμές από την καλούσα συνάρτηση.

Προαναγγελία πρωτότυπου συνάρτησης, όταν καλείται πριν ορισθεί

```
<Τύπος Επιστροφής> <Όνομα Συνάρτησης> (<Τυπικές Παράμετροι>);
```

```
int main(void)  
{  
    ....  
}
```

```
<Τύπος Επιστροφής> <Όνομα Συνάρτησης> (<Τυπικές Παράμετροι>)
```

```
{  
    ....  
}
```

<sup>1</sup> Εικάζεται, αλλά δεν έχει αποδειχθεί μαθηματικά, ότι αν ξεκινήσουμε από ένα θετικό ακέραιο αριθμό  $n$  και πάρουμε σαν επόμενο του τον  $n/2$ , αν ο  $n$  είναι άρτιος, ή τον  $3n+1$ , αν ο  $n$  είναι περιττός, συνεχίζοντας με αυτόν τον τρόπο, κάποια στιγμή θα καταλήξουμε στο 1. Για παράδειγμα, αν ξεκινήσουμε από το  $n=22$ , η ακολουθία αριθμών που θα πάρουμε με αυτή τη διαδικασία θα είναι η 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

**1.3** Δομήστε το πρόγραμμά σας σε ανεξάρτητα<sup>2</sup> αρχεία. Για να διαβάσετε τον ακέραιο  $n$  στη συνάρτηση `main`, χρησιμοποιήστε τη συνάρτηση `int getinteger(int base)`, που θα βρείτε στο πρόγραμμα <http://www.di.uoa.gr/~ip/cprogs/convdecbin.c> (σελ. 71 σημειώσεων).

<code>main.c</code>	<code>isodd.c</code> Κώδικας συνάρτησης	<code>isodd.h</code> Πρωτότυπο συνάρτησης
<pre>#include &lt;stdio.h&gt; #include "isodd.h" #include "getinteger.h"  int main(void) {     ..... }</pre>	<pre>int isodd(int n) {     ..... }</pre>	<pre>int isodd(int);</pre>

<code>getinteger.c</code> Κώδικας συνάρτησης	<code>getinteger.h</code> Πρωτότυπο συνάρτησης και ορισμός συμβολικής σταθεράς
<pre>#include &lt;stdio.h&gt; #include "getinteger.h"  int getinteger(int base) {     ..... }</pre>	<pre>#define ERROR -1  int getinteger(int);</pre>

**1.3.1** Μεταγλωττίστε το πηγαίο αρχείο `isodd.c` για να παραγάγετε το αντικειμενικό αρχείο `isodd.o`

**1.3.2** Μεταγλωττίστε το πηγαίο αρχείο `getinteger.c` για να παραγάγετε το αντικειμενικό αρχείο `getinteger.o`

**1.3.3** Μεταγλωττίστε το πηγαίο αρχείο `main.c` για να παραγάγετε το αντικειμενικό αρχείο `main.o`

**1.3.4** Συνδέστε τα αντικειμενικά αρχεία `main.o`, `isodd.o` και `getinteger.o` για να παραγάγετε το εκτελέσιμο αρχείο `myprog`.

**Άσκηση 2:** Υπολογισμός όρων ακολουθίας Fibonacci.

**2.1:** Η ακολουθία Fibonacci ορίζεται από τη συνάρτηση:

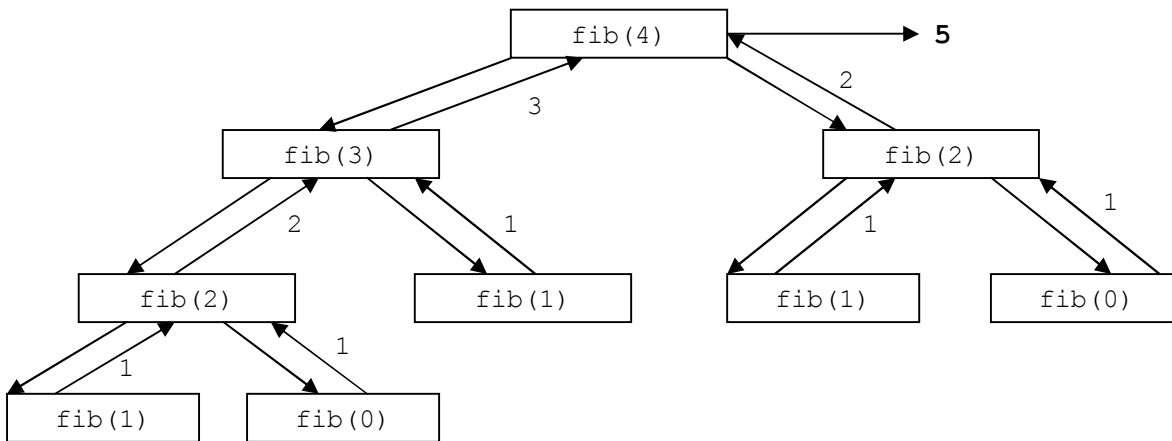
$$f(n) = \begin{cases} 1 & n = 0, n = 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

Ορίστε την αναδρομική συνάρτηση `int fib(int n)` που να υλοποιεί τον υπολογισμό του  $n$ -οστού όρου της ακολουθίας Fibonacci.

<sup>2</sup> Για τη δημιουργία σε Dev-C++ ενός project, με σκοπό τη διάσπαση ενός προγράμματος σε αρχεία σύμφωνα με τη δόμηση που εμφανίζεται παραπάνω, δείτε το παράρτημα.

**2.2** Δημιουργήστε το πηγαίο αρχείο `recfib.c` το οποίο να υπολογίζει τους όρους της ακολουθίας Fibonacci από  $n=28$  μέχρι το  $n=35$ , καλώντας τη συνάρτηση που υλοποιήσατε στο 2.1. Αμέσως μετά τον υπολογισμό, να εμφανίζεται η τιμή του αντίστοιχου όρου της ακολουθίας Fibonacci στην οθόνη.


**2.3** Μετρήστε τις αναδρομικές κλήσεις για κάθε εκτέλεση του παραπάνω προγράμματος χρησιμοποιώντας μία καθολική (global) μεταβλητή. Εκτυπώστε το πλήθος των αναδρομικών κλήσεων μετά από κάθε εκτέλεση.



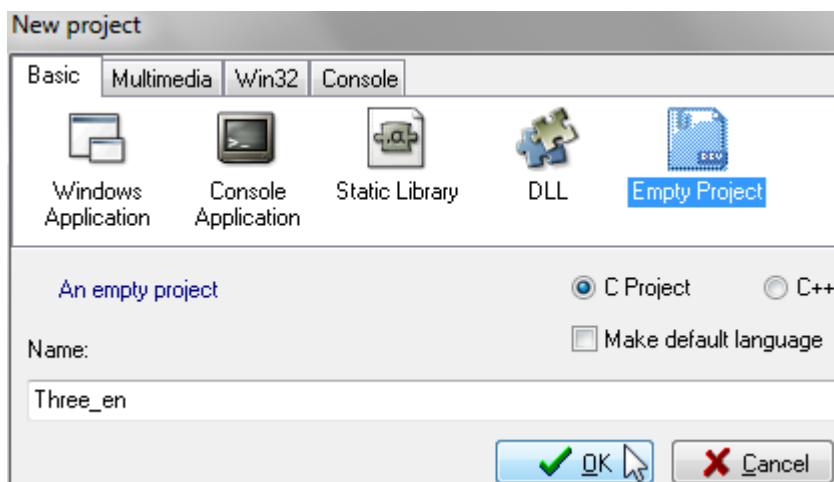
**2.4** Παρατηρήστε το δένδρο αναδρομικών κλήσεων του προγράμματος για  $n=4$  για εξηγήστε την αύξηση του χρόνου υπολογισμού του προγράμματος σε σχέση με τον όρο που υπολογίζεται.

**2.5** Ο  $n$ -οστός όρος της ακολουθίας Fibonacci μπορεί να υπολογισθεί και επαναληπτικά, αν χρησιμοποιήσουμε δύο μεταβλητές για να αποθηκεύουμε σε κάθε βήμα τους δύο προηγούμενους αριθμούς, ώστε προσθέτοντας τους, να υπολογίζουμε τον επόμενο. Κατασκευάστε το πρόγραμμα `fibonacci.c` που να υλοποιεί την παραπάνω επαναληπτική διαδικασία.

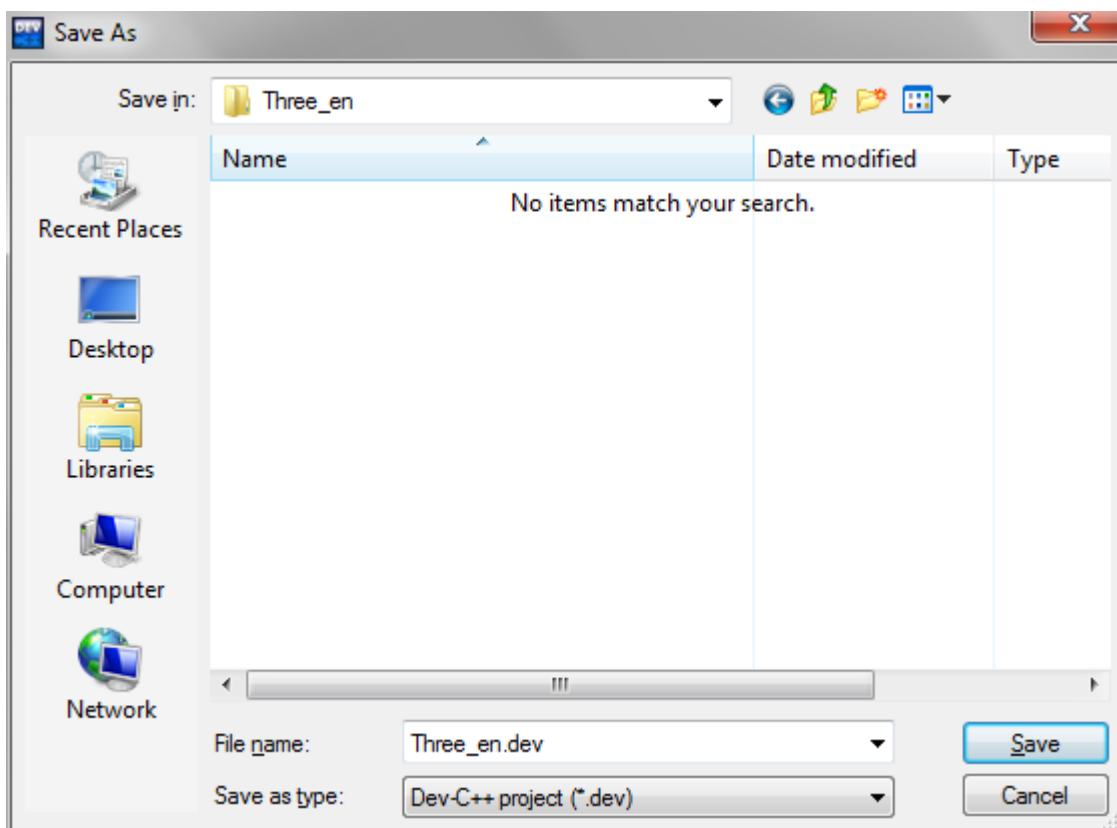
## ΠΑΡΑΡΤΗΜΑ: Δημιουργία projects σε Dev-C++

Δημιουργούμε νέο project επιλέγοντας από το μενού File→New→Project, ή, εναλλακτικά, πατώντας το εικονίδιο .

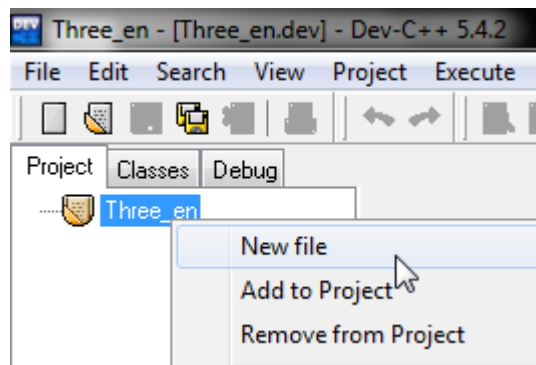
Στην οθόνη που εμφανίζεται, επιλέγουμε “Empty Project”. Για να ορίσουμε ότι θα γράψουμε σε γλώσσα C, τσεκάρουμε την επιλογή “C Project”. Τέλος, επιλέγουμε ένα όνομα για το project και πατάμε το OK.



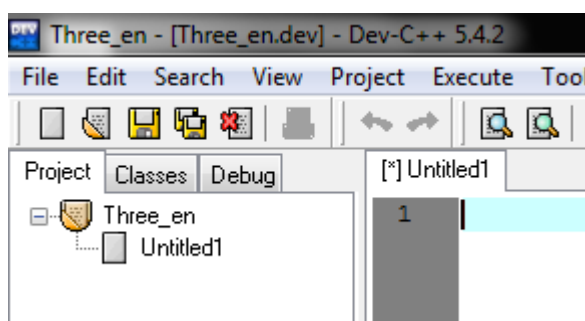
Εμφανίζεται η οθόνη αποθήκευσης του project στον σκληρό δίσκο, οπότε επιλέγουμε έναν κατάλογο και πατάμε το “Save”.




Για να προσθέσουμε νέα αρχεία κώδικα στο project μας, κάνουμε δεξί κλικ στο όνομα του project στο αριστερό μέρος της οθόνης και επιλέγουμε “New File”.



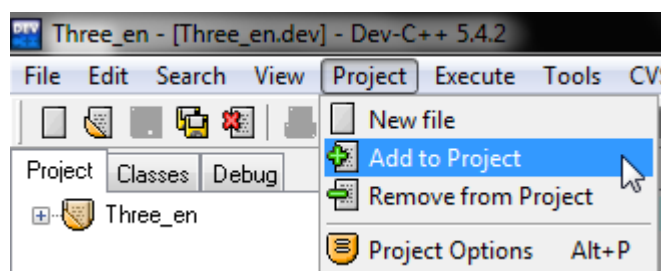
ή εναλλακτικά επιλέγουμε από το μενού “Project”→“New File” οπότε και εμφανίζεται νέα, κενή καρτέλα.



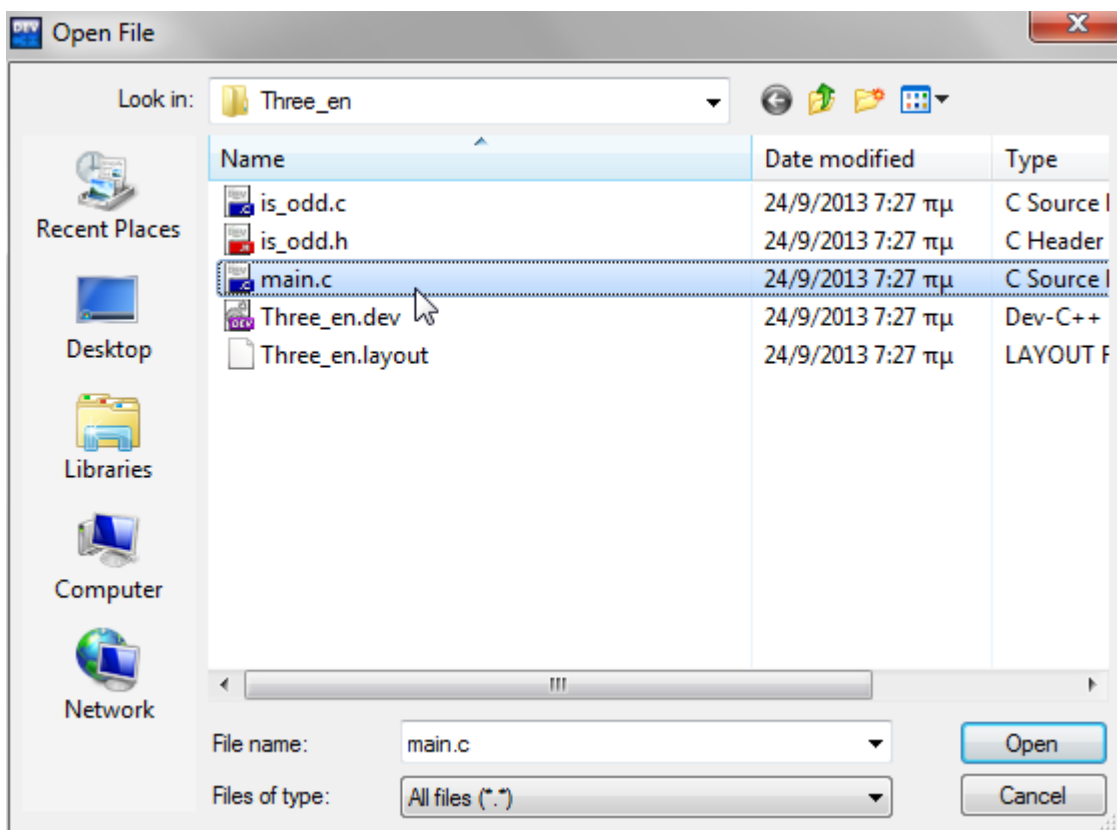
Αφού ολοκληρώσουμε, πατάμε Ctrl+S ή File→Save ή επιλέγουμε το εικονίδιο . Εμφανίζεται η οθόνη αποθήκευσης του αρχείου κώδικα στη οποία πληκτρολογούμε το όνομα του αρχείου μαζί με την επέκτασή του (.c ή .h).

Την ίδια διαδικασία ακολουθούμε για να προσθέσουμε στο project και οποιοδήποτε άλλο νέο αρχείο κώδικα (\*.c) ή αρχείο επικεφαλίδας (\*.h) επιθυμούμε.



Επίσης, μας παρέχεται η δυνατότητα να προσθέσουμε ένα αρχείο κώδικα που ήδη έχουμε δημιουργήσει και αποθηκεύσει στον υπολογιστή μας, επιλέγοντας από το μενού “Project”→ “Add To Project, ή το σχετικό κουμπί από τη μπάρα εικονιδίων.



Εμφανίζεται η οθόνη επιλογής του αρχείου:



στην οποία επιλέγουμε το αρχείο και το ενσωματώνουμε κάνοντας κλικ στο Open.

Αφού ολοκληρώσουμε τη δημιουργία των αρχείων κώδικα ή την προσθήκη αρχείων που ήδη έχουμε αποθηκευμένα στον υπολογιστή μας, πατάμε το κουμπί  για να μεταγλωττίσουμε τα αρχεία κώδικα (όλα μαζί) και τέλος το κουμπί  για να εκτελέσουμε το πρόγραμμά μας.

Κατά την παραπάνω διαδικασία, αποθηκεύσαμε και ένα αρχείο με κατάληξη .dev στον υπολογιστή μας. Το αρχείο αυτό περιέχει όλες τις πληροφορίες ενσωμάτωσης πηγαίων αρχείων στο project μας και είναι αυτό που ανοίγουμε από το μενού "File" → "Open" ώστε να συνεχίσουμε την εργασία μας σε ένα project που έχουμε ήδη δημιουργήσει.

## ΕΡΓΑΣΤΗΡΙΟ 7: Δείκτες και Πίνακες

Στο εργαστήριο αυτό, θα μάθουμε για τους δείκτες της C και θα τους χρησιμοποιήσουμε για να κατασκευάσουμε συναρτήσεις που επιστρέφουν τιμές, μέσω ορισμάτων που είναι δείκτες, στις συναρτήσεις που τις καλούν. Επίσης, θα μάθουμε να ορίζουμε και να χρησιμοποιούμε πίνακες για να υλοποιούμε περισσότερο πολύπλοκες αλγοριθμικές διαδικασίες, θα δούμε τη σχέση τους με τους δείκτες και πώς να τους αντιμετωπίζουμε σαν τύπους δεδομένων (για να τους περνάμε σαν ορίσματα σε συναρτήσεις, κλπ.).

### Άσκηση 1: Πέρασμα δεδομένων μέσω δεικτών

1.1 Γράψτε την παρακάτω συνάρτηση μέσα σ' ένα αρχείο `myfun.c` και μεταγλωττίστε το:

```
void badf(int x, int y, int sum, int diff)
{
    sum = x+y;
    diff = x-y;
}
```

Γράψτε και μία συνάρτηση `main` μέσα σ' ένα αρχείο `myprog.c` η οποία να υλοποιεί την παρακάτω διαδικασία (εκφρασμένη σε ψευδογλώσσα) και μεταγλωττίστε το:

```
Όρισε τις ακέραιες μεταβλητές a, b, sum, diff
Θέσε a=5, b=4, sum=0, diff=0
Κάλεσε badf(a, b, sum, diff)
Τύπωσε sum, diff
```

Δημιουργήστε το εκτελέσιμο πρόγραμμα `myprog` (από τα αντικειμενικά αρχεία `myfun.o` και `myprog.o`) και εκτελέστε το. Τι παρατηρείτε; Μπορείτε να εξηγήσετε το αποτέλεσμα;

1.2 Τροποποιήστε τη συνάρτηση `void badf(int x, int y, int sum, int diff)` ώστε οι `sum` και `diff` να είναι δείκτες σε ακεραίους, και μετονομάστε την σε `goodf`. Τροποποιήστε την `main` του προγράμματος ώστε να καλεί τη συνάρτηση `goodf` αντί για την `badf`. Τι παρατηρείτε;

1.3 Χρησιμοποιήστε τη συνάρτηση `scanf()` για να διαβάσετε τους ακέραιους αριθμούς `a`, `b` που εμφανίζονται στην `main`.

Η συνάρτηση `scanf()` διαβάζει από την είσοδο δεδομένα προς επεξεργασία. Συντάσσεται με αντίστοιχο τρόπο με την `printf()` δηλαδή:

```
scanf("%y", &var);
```

όπου `y` είναι σύμβολο που αντιστοιχεί στον τύπο δεδομένων της μεταβλητής `var` (π.χ, `d` για `int`, `f` για `float`, κλπ.). Παρατηρούμε ότι στο δεύτερο όρισμα περνιέται η διεύθυνση της μεταβλητής `var`, για να αποθηκευθεί η καταχώρηση και μετά το πέρας της κλήσης.

## Άσκηση 2: Το κόσκινο του Ερατοσθένη

Το κόσκινο του Ερατοσθένη είναι ένας αλγόριθμος για την εύρεση όλων των πρώτων αριθμών σε ένα εύρος τιμών (από 2 έως  $N$ ). Είναι ένας από τους αρχαιότερους γνωστούς αλγόριθμους και οφείλεται στον Έλληνα φιλόσοφο και αστρονόμο Ερατοσθένη (276-194 π.Χ.). Ο αλγόριθμος εξετάζει διαδοχικά όλους τους ακεραίους και για κάθε αριθμό που συναντά διαγράφει όλα τα πολλαπλάσιά του (αφού σίγουρα δεν είναι πρώτοι).

Παρατηρήστε τα τρία πρώτα βήματα του αλγορίθμου, για  $N=20$ .

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Ο «2» είναι πρώτος, διαγραφή των 4, 6, 8, 10, 12, 14, 16, 18, 20.																		
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Ο «3» είναι πρώτος, διαγραφή των 6, 9, 12, 15, 18.																		
Ο «4» δεν εξετάζεται γιατί έχει αφαιρεθεί σε προηγούμενο βήμα.																		

Ορίστε έναν πίνακα  $N$  θέσεων και αρχικοποιήστε τον με μονάδες. Η διάσταση του πίνακα να ορίζεται μέσω `#define` με τιμή ίση με 50. Υλοποιήστε το κόσκινο του Ερατοσθένη σύμφωνα με τον αλγόριθμο όπως εκφράζεται παρακάτω σε ψευδογλώσσα:

```
Για i=2 έως N-1 επανάλαβε
    Θέσε A[i]=1

Για i=2 έως N-1 επανάλαβε
    Αν A[i]!=0
        Για j=2*i έως N-1 με βήμα i επανάλαβε
            Θέσε A[j]=0

Για i=2 έως N-1 επανάλαβε
    Αν το A[i]==1 τύπωσε "ο i είναι πρώτος"
```

Εκτελέστε το πρόγραμμά σας και επιβεβαιώστε την ορθότητα του αποτελέσματος.



### Άσκηση 3: Πίνακες και αριθμητική δεικτών

**3.1** Ορίστε τη συνάρτηση `void print_array(int *A, int n)` που δέχεται σαν όρισμα έναν πίνακα ακεραίων και τη διάστασή του και εκτυπώνει τα περιεχόμενα του σε μία γραμμή χωρισμένα με στηλογνόμωνα (`tab`).

**3.2** Δημιουργήστε το πρόγραμμα `pointers.c` που δημιουργεί έναν πίνακα 8 θέσεων και κάνει πράξεις πάνω σε αυτόν, ακολουθώντας το παρακάτω τμήμα κώδικα:

```
01: int i, a[8], *pa;
02:
03: for (i=0; i<8; i++)
04:   a[i] = i*i;
05:
06: pa = &a[0];
07: a[6] = *(a+4);
08: *(pa+3) = a[5];
09: a[0] = *((pa++)+2);
10: *((++pa)+5) = a[1];
11: *(&a[5]-1) = * (--pa);
```

**3.3** Εκτυπώστε τα περιεχόμενα του πίνακα, χρησιμοποιώντας τη συνάρτηση `print_array()` μετά από τα βήματα 04, 07, 08, 09, 10 και 11 για να παρακολουθήσετε την επίδραση των εντολών στον πίνακα.

### Άσκηση 4: Πίνακες και συναρτήσεις

Σε ένα αγώνισμα υπάρχουν 10 κριτές. Η βαθμολογία του κάθε αγωνιζόμενου βγαίνει από τον μέσο όρο 8 κριτών, αφού εξαιρεθεί αυτός με τη μεγαλύτερη και αυτός με τη μικρότερη βαθμολογία (για να αποφευχθούν φαινόμενα μεροληψίας, είτε υπέρ είτε κατά ενός αθλητή).

**4.1** Ορίστε τη συνάρτηση `int max_array(int *A, int n)` που επιστρέφει στο όνομά της το μέγιστο στοιχείο του πίνακα ακεραίων `A` διάστασης `n`.

**4.2** Ορίστε τη συνάρτηση `int min_array(int *A, int n)` που επιστρέφει στο όνομά της το ελάχιστο στοιχείο του πίνακα ακεραίων `A` διάστασης `n`.

**4.3** Ορίστε τη συνάρτηση `int sum_array(int *A, int n)` που επιστρέφει στο όνομά της το άθροισμα των στοιχείων του πίνακα ακεραίων `A` διάστασης `n`.

**4.4** Κατασκευάστε το πρόγραμμα `judgement.c` το οποίο να διαβάζει τις 10 βαθμολογίες με χρήση της `scanf()` και να υπολογίζει τη βαθμολογία σύμφωνα με τον τύπο:

$$\text{Βαθμός} = (\text{Άθροισμα} - \text{Μέγιστο} - \text{Ελάχιστο}) / 8$$



## ΕΡΓΑΣΤΗΡΙΟ 8: Πολυδιάστατοι Πίνακες και Δυναμική Δέσμευση Μνήμης

Στο εργαστήριο αυτό θα μελετήσουμε τον τρόπο με τον οποίο ορίζουμε στην C πολυδιάστατους πίνακες και θα δούμε πώς μπορούμε να δεσμεύουμε δυναμικά μνήμη για να δημιουργούμε πίνακες, όταν δεν γνωρίζουμε κατά τη φάση συγγραφής του προγράμματος τις διαστάσεις των πινάκων που χρειαζόμαστε.

### Άσκηση 1: Διδιάστατοι πίνακες

**1.1** Κατασκευάστε το πρόγραμμα `twodim.c` που ορίζει στατικά έναν πίνακα `A` διαστάσεων  $6 \times 10$  και αρχικοποιεί το στοιχείο `A[i][j]` που βρίσκεται στη γραμμή `i` και στη στήλη `j`, σύμφωνα με τον τύπο:

$$A[i][j] = i(5 - i) + j(9 - j)$$

Εκτυπώστε τον πίνακα κατά γραμμές, με τα στοιχεία κάθε γραμμής χωρισμένα με στηλογνώμονα (`tab`).

**1.2** Επεκτείνετε το πρόγραμμα `twodim.c` ώστε να εκτυπώνει τον ανάστροφο του πίνακα, δηλαδή αυτόν που έχει στήλες τις γραμμές του αρχικού και γραμμές τις στήλες του αρχικού.

<p>Ο αρχικός πίνακας:</p> <pre>1 4 6 7 8 5 6 7 1 3 1 0 4 7 2</pre>	<p>Ο ανάστροφος πίνακας:</p> <pre>1 5 1 4 6 0 6 7 4 7 1 7 8 3 2</pre>
--	---

**1.3** Επεκτείνετε το πρόγραμμα `twodim.c` ώστε να εκτυπώνει τις γραμμές του αρχικού πίνακα με αντίστροφη σειρά.

<p>Ο αρχικός πίνακας:</p> <pre>1 4 6 7 8 5 6 7 1 3 1 0 4 7 2</pre>	<p>Εκτύπωση γραμμών με αντίστροφη σειρά:</p> <pre>8 7 6 4 1 3 1 7 6 5 2 7 4 0 1</pre>
--	---

**1.4** Επεκτείνετε το πρόγραμμα `twodim.c` ώστε να εκτυπώνει όλα τα στοιχεία του πίνακα σε μία σειρά, διασχίζοντάς τον με ένα «φιδοειδή» τρόπο, όπως φαίνεται στο σχήμα:

<p>Ο αρχικός πίνακας:</p> <pre>1 4 6 7 8 5 6 7 1 3 1 0 4 7 2</pre>	<p>«Φιδοειδής» εκτύπωση των στοιχείων:</p> <pre>1 4 6 7 8 3 1 7 6 5 1 0 4 7 2</pre>
--	---

**Άσκηση 2:** Δυναμική δέσμευση μνήμης για μονοδιάστατο πίνακα

**2.1** Κατασκευάστε το πρόγραμμα `array.c` που να διαβάζει από την είσοδο τη διάσταση ενός μονοδιάστατου πίνακα ακεραίων (έστω  $N$ ), να δεσμεύει χώρο  $N$  θέσεων δυναμικά και έπειτα να τον αρχικοποιεί διαβάζοντας από την είσοδο ακέραιους αριθμούς.

**Δυναμική δέσμευση μνήμης για μονοδιάστατο πίνακα.**

Συνάρτηση `void *malloc(unsigned int size)`

Χρήση:

```
TΔ *p; //Δηλωση ενός δεικτη σε στοιχεία τυπου TΔ  
p = malloc(N * sizeof(TΔ)); //Δεσμευση μνημης για N στοιχεία τυπου TΔ
```

Η `malloc` επιστρέφει `NULL` σε περίπτωση αποτυχίας δέσμευσης της αιτούμενης μνήμης.

**Αποδέσμευση μνήμης για δυναμικά δεσμευμένους πίνακες.**

Συνάρτηση `void free(void *p)`

Χρήση: `free(p)`;

όπου `p` είναι δείκτης σε θέσεις μνήμης που έχουν δεσμευθεί δυναμικά.

Όταν χρησιμοποιούνται οι συναρτήσεις `malloc()` και `free()`, πρέπει να γίνεται συμπερίληψη του αρχείου επικεφαλίδας `stdlib.h`.

Εκτυπώστε τα στοιχεία του πίνακα σε μία γραμμή χωρισμένα με στηλογνώμονα (`tab`).

**2.2** Κατασκευάστε το αρχείο `input.txt` το οποίο να περιέχει  $N+1$  ακέραιους αριθμούς ως εξής: Ο πρώτος αριθμός είναι το πλήθος των στοιχείων ( $N$ ) και ακολουθούν οι  $N$  ακέραιοι χωρισμένοι με κενά.

**2.3** Εκτελέστε το πρόγραμμα `array` με ανακατεύθυνση εισόδου από το αρχείο `input.txt`.

**2.4** Επεκτείνετε το πρόγραμμά σας, ώστε να εκτυπώνεται ο μέσος όρος των στοιχείων του πίνακα.

**Άσκηση 3:** Δυναμική δέσμευση μνήμης για διδιάστατο πίνακα

**3.1** Κατασκευάστε το πρόγραμμα `mines.c` που να διαβάζει από την είσοδο τις διαστάσεις ενός διδιάστατου πίνακα χαρακτήρων (έστω  $N \times M$ ), να δεσμεύει χώρο  $N \times M$  θέσεων δυναμικά και έπειτα να τον αρχικοποιεί διαβάζοντας από την είσοδο χαρακτήρες.

**Δυναμική δέσμευση μνήμης για διδιάστατο πίνακα διάστασης  $N \times M$** 

```
TΔ **p;
P = malloc(N * sizeof(TΔ *));
for (i = 0 ; i < N ; i++)
    p[i] = malloc(M * sizeof(TΔ));
```

**Αποδέσμευση μνήμης για δυναμικά δεσμευμένο πίνακα  $N \times M$** 

```
for (i = 0 ; i < N ; i++)
    free(p[i]);
free(p);
```

**3.2** Κατασκευάστε το αρχείο `mines.txt` που αναπαριστά τα δεδομένα ενός ναρκοπεδίου. Συγκεκριμένα, στην πρώτη γραμμή του αρχείου υπάρχουν οι διαστάσεις του ναρκοπεδίου ( $N$  και  $M$ ) και μετά ακολουθούν γραμμή-γραμμή τα περιεχόμενα των κελιών, που είναι ο χαρακτήρας `'.'` όταν δεν υπάρχει νάρκη και ο χαρακτήρας `'*'` όταν υπάρχει νάρκη.

```
3 4
..*.
.**.
*.*.
```

**3.3** Επεκτείνετε το πρόγραμμά σας, ώστε να εκτυπώνονται τα περιεχόμενα του πίνακα που διαβάστηκε. Εκτελέστε το πρόγραμμά σας με ανακατεύθυνση εισόδου από το αρχείο `mines.txt`.

**3.4** Επεκτείνετε το πρόγραμμά σας, ώστε να εκτυπώνεται μια τροποποιημένη μορφή του ναρκοπεδίου, στην οποία, στα κελιά που υπάρχει νάρκη να εμφανίζεται πάλι το `'*'`, ενώ στα κελιά που δεν υπάρχει νάρκη να φαίνεται ένας αριθμός που δείχνει σε πόσα γειτονικά κελιά υπάρχει νάρκη. Σαν γειτονικά θεωρούνται όχι μόνο συνεχόμενα οριζόντια ή κατακόρυφα κελιά, αλλά και συνεχόμενα σε διαγώνια κατεύθυνση. Για παράδειγμα, για το ναρκοπέδιο που είδαμε, θα πρέπει να εμφανίζεται η έξοδος:

```
13*2
2**3
*4*2
```

**ΠΑΡΑΡΤΗΜΑ:** Αποσφαλμάτωση προγραμμάτων (Πράξη 3<sup>η</sup>)

Στα εργαστήρια 3 και 4 είχαμε αναφερθεί στα συντακτικά και λογικά λάθη που μπορεί να έχουν τα προγράμματά μας. Στον προγραμματισμό υπάρχει και ένα ακόμα σημαντικό είδος σφαλμάτων, τα λάθη διαχείρισης μνήμης. Η ύπαρξή τους στη γλώσσα προγραμματισμού C οφείλεται, σε μεγάλο βαθμό, στην ελευθερία που δίνεται στον προγραμματιστή μέσω των δεικτών, ένα πολύ ισχυρό εργαλείο, που όμως πρέπει να χειριζόμαστε με προσοχή. Στο σημερινό εργαστήριο θα δούμε τι εννοούμε όταν αναφερόμαστε σε σφάλματα διαχείρισης μνήμης, καθώς και πώς μπορούμε να τα ανιχνεύουμε και να τα διορθώνουμε με τη βοήθεια του debugger gdb, που συνήθως είναι εγκατεστημένος σε συστήματα Unix/Linux.

**Σφάλματα διαχείρισης μνήμης**

Ένα σφάλμα διαχείρισης μνήμης συνήθως το συνδέουμε στο μυαλό μας με την εμφάνιση του μηνύματος “Segmentation Fault” (ή ενός παραθύρου για το κλείσιμο του προγράμματος αν το τρέχουμε μέσα από το Dev C++). Γενικά, τα σφάλματα διαχείρισης μνήμης έχουν να κάνουν με το λανθασμένο χειρισμό κάποιας περιοχής της μνήμης, που ενώ νομίζουμε ότι περιέχει κάτι (π.χ. έναν πίνακα) και προσπελάζουμε αυτές τις θέσεις μνήμης σαν αυτό το κάτι να ήταν εκεί, τελικά δεν υπάρχει αυτό που νομίζαμε.

Το πιο συνηθισμένο σφάλμα διαχείρισης μνήμης είναι να προσπελάσουμε το περιεχόμενο ενός δείκτη, χωρίς αυτός να δείχνει σε δεσμευμένες θέσεις μνήμης ή να προσπελάσουμε θέσεις μνήμης πέρα απ' αυτές που έχουμε δεσμεύσει. Ας δούμε κάποια σχετικά παραδείγματα.

```
#include <stdio.h>

int main(void) {
    int *p;
    *p = 10;
    return 0;
}
```

Στο προηγούμενο παράδειγμα, προσπαθούμε να βάλουμε στις θέσεις μνήμης που δείχνει ο `p` τον αριθμό 10. Όμως, δεν έχουμε δεσμεύσει χώρο, στον οποίο θα δείχνει το `p`, ικανό να χωρέσει έναν ακέραιο. Άρα, το προηγούμενο παράδειγμα, κατά πάσα πιθανότητα, θα κάνει segmentation fault.

```
#include <stdio.h>

int main(void) {
    int array[10];
    array[10] = 5;
    return 0;
}
```

Εδώ, πηγαίνουμε και προσπελάζουμε την ενδέκατη θέση του πίνακα `array`, ενώ ο πίνακας έχει μόνο 10 θέσεις (θυμηθείτε ότι η πρώτη θέση ενός πίνακα είναι η 0). Αν και αυτό είναι ένα προφανές λάθος διαχείρισης μνήμης, προσπαθήστε να τρέξετε το πρόγραμμα. Η εκτέλεσή του έγινε ομαλά; Γιατί πιστεύετε ότι συνέβη αυτό;

Ενώ είναι εφικτό η ανίχνευση των σφαλμάτων διαχείρισης μνήμης να γίνει με εξαντλητική ιχνηλάτηση του κώδικα και χρήση `printf`, ο τρόπος αυτός είναι πολύ αναποτελεσματικός και κουραστικός. Γι' αυτό το λόγο, αλλά και για την ευκολότερη ανίχνευση και των λογικών λαθών στα οποία αναφερθήκαμε στο εργαστήριο 4, θα δούμε στη συνέχεια πώς δουλεύει ένα εργαλείο

αποσφαλμάτωσης (debugging), ο gdb Το εργαλείο αυτό θα κάνει την ανίχνευση και τη διόρθωση των λογικών λαθών και των σφαλμάτων διαχείρισης μνήμης πολύ πιο εύκολη, ενώ δεν θα απαιτεί να πειράξουμε τον κώδικα για την εισαγωγή `printf`.

## O debugger gdb

Ο debugger gdb είναι εγκατεστημένος στα συστήματα Unix και Linux της σχολής. Για να τον χρησιμοποιήσουμε, δίνουμε σαν παράμετρο το `-g3` κατά τη μεταγλώττιση του προγράμματος, π.χ.

```
gcc -g3 -o my_prog my_prog.c
```

Γράφοντας `gdb ./my_prog` ξεκινά η εκτέλεση του debugger, οπότε και εμφανίζεται μία γραμμή εντολών. Οι επιλογές που έχουμε στη διάθεσή μας είναι οι εξής (στις παρενθέσεις αναφέρονται τα συντεταγμένα ονόματα των εντολών):

### **break** όνομα\_συνάρτησης (b)

Με αυτή την εντολή, η εκτέλεση του προγράμματος θα ανασταλεί όταν γίνει η πρώτη εκτέλεση της συνάρτησης που καθορίσαμε. Αν γράψουμε π.χ. `b main`, τότε η εκτέλεση του προγράμματος θα ανασταλεί αμέσως μόλις αυτό ξεκινήσει.

### **break** γραμμή (b)

Με αυτή την εντολή, θέτουμε ένα σημείο διακοπής (breakpoint) σε συγκεκριμένη γραμμή, οπότε η εκτέλεση του προγράμματος θα ανασταλεί μόλις ο έλεγχος φτάσει στη γραμμή που δώσαμε. Τα breakpoints δεν μπορούν να μπουν σε γραμμές που είναι κενές ή έχουν μόνο σχόλια. Η εισαγωγή τουλάχιστον ενός breakpoint είναι απαραίτητη, γιατί αλλιώς δεν θα μπορούσαμε να εκτελέσουμε βηματικά τον κώδικά μας.

### **run** όρισμα<sub>1</sub> όρισμα<sub>2</sub> ... όρισμα<sub>n</sub> (r)

Αφότου έχουμε βάλει τουλάχιστον ένα breakpoint, δίνουμε αυτή την εντολή για να ξεκινήσει η εκτέλεση του προγράμματος (μέχρι να φτάσει στο πρώτο breakpoint). Αν το πρόγραμμά μας παίρνει ορίσματα από τη γραμμή εντολής, τα δίνουμε εδώ, αλλιώς γράφουμε απλά `r`.

### **step** (s)

Όταν η εκτέλεση του προγράμματος έχει ανασταλεί, μπορούμε να συνεχίσουμε την εκτέλεση βηματικά, δηλαδή να εκτελείται μόνο μία γραμμή κώδικα κάθε φορά. Με την εντολή `s` εκτελείται η τρέχουσα γραμμή κώδικα, ενώ αν αυτή είναι η κλήση κάποια συνάρτησης, ο έλεγχος μεταφέρεται εντός της.

### **next** (n)

Το ίδιο με την `s`, μόνο που αν συναντήσει συνάρτηση την εκτελεί ολόκληρη χωρίς να μπει μέσα, οπότε ο έλεγχος μεταφέρεται στη γραμμή κώδικα μετά την κλήση της συνάρτησης.

### **finish** (f)

Με αυτή την εντολή εκτελείται μέχρι τέλους η τρέχουσα συνάρτηση και η βηματική εκτέλεση συνεχίζει μέσα στην συνάρτηση που την κάλεσε.

### **print** παράσταση (p)

Με αυτή την εντολή, εμφανίζεται η τιμή της παράστασης που δίνουμε, με βάση τις τρέχουσες τιμές των μεταβλητών. Η παράσταση μπορεί να είναι κάτι περίπλοκο, όπως `array[x]+y`, ή απλά μια μεταβλητή π.χ. `y` ή `array[2]`.

### **continue** (c)

Η εκτέλεση του κώδικα συνεχίζεται κανονικά χωρίς βηματική εκτέλεση.

**backtrace (bt)**

Μας εμφανίζει τη στοίβα των συναρτήσεων. Με αυτή την εντολή, μπορούμε να δούμε ποια συνάρτηση κάλεσε την τρέχουσα συνάρτηση, ποια κάλεσε αυτή κλπ.

**quit (q)**

Σταματά η εκτέλεση του gdb και επιστρέφουμε στη γραμμή εντολής.

Πέρα από τις προφανείς ευκολίες που παρέχει ένας debugger, αν χειριστούμε σωστά τις προηγούμενες εντολές, έχει και τη δυνατότητα να μας δώσει υπερπολύτιμες πληροφορίες, όταν το πρόγραμμά μας κάνει segmentation fault. Ας το δούμε αυτό με ένα παράδειγμα.

```
#include <stdio.h>

int main(void) {
    int p[] = {10, -1, 8, 6, 9, 13, 0, -9, 6};
    int count = 0, sum = 0, i = 0;
    do {
        if (p[i] > 0) {
            count++;
            sum += p[count];
        }
        i++;
    } while (1);
    printf("There are %d positive numbers with sum %d\n", count, sum);
    return 0;
}
```

Το προηγούμενο πρόγραμμα προσπαθεί να μετρήσει πόσοι αριθμοί στον πίνακα `p` είναι θετικοί και να υπολογίσει το άθροισμά τους (παρατηρήστε ότι το μέγεθος του πίνακα δεν είναι καθορισμένο με άμεσο τρόπο).

Τρέχοντας αυτό το πρόγραμμα, αργά η γρήγορα θα μας δώσει segmentation fault. Ας τρέξουμε λοιπόν τον gdb για να μας βοηθήσει.

Αρχικά, μεταγλωττίζουμε το πρόγραμμα γράφοντας

```
gcc -g3 -o my_prog my_prog.c
```

όπου `my_prog.c` το όνομα που έχουμε δώσει στο αρχείο με τον πηγαίο κώδικα.

Γράφουμε `gdb ./my_prog` για να αρχίσει η εκτέλεση του gdb.

Τώρα δεν έχουμε παρά να δώσουμε `r` και να αφήσουμε τον gdb να τρέξει το πρόγραμμα. Κάποια στιγμή θα μας δώσει ένα μήνυμα που θα μοιάζει με αυτό

```
Program received signal SIGSEGV, Segmentation fault.
0x080483ac in main () at my_prog.c:7
7          if (p[i] > 0) {
```

το οποίο μας λέει όχι μόνο ότι υπήρξε segmentation fault, αλλά και σε ποια γραμμή ποιας συνάρτησης (`in main () at my_prog.c:7`) εμφανίστηκε, ενώ τυπώνει και το περιεχόμενο αυτής της γραμμής `if (p[i] > 0)`.



Οπότε, ο gdb μας βοήθησε να ανιχνεύσουμε πού συμβαίνει αυτό το λάθος διαχείρισης μνήμης. Ας δούμε πώς μπορεί να μας βοηθήσει να το διορθώσουμε.

Παρατηρήστε ότι η εκτέλεση του προγράμματος δεν έχει σταματήσει. Ο gdb μπορεί ακόμα να δεχτεί εντολές. Αυτό σημαίνει ότι μπορούμε να τον ρωτήσουμε για τις τιμές οποιασδήποτε μεταβλητής θέλουμε.

Αφού λοιπόν, το πρόβλημα έγινε στη γραμμή `if (p[i] > 0)`, ας επικεντρωθούμε σε όσες μεταβλητές περιλαμβάνει.

```
p p[i]
Cannot access memory at address 0xbf8de000
```

Άρα το `p[i]` αναφέρεται σε κάποια περιοχή της μνήμης στην οποία δεν έχουμε πρόσβαση, άρα σωστά το πρόγραμμά μας έκανε `segmentation fault`. Συνεπώς, αυτό σημαίνει ότι έχουμε βγει έξω από τα όρια του πίνακα `p`. Ας δούμε πού έχουμε φτάσει

```
p i
$3 = 1279
```

Το `$3` δεν πρέπει να σας απασχολεί. Αυτό που μας ενδιαφέρει είναι η αριθμητική τιμή που εμφανίζεται, το 1279. Άρα το `i` έχει φτάσει στο 1279! Είναι προφανές ότι κάπου μέσα στο πρόγραμμά μας έχουμε ξεχάσει να κάνουμε έλεγχο για να μην ξεπερνάμε τα όρια του πίνακα `p`. Όντως, πουθενά στον κώδικά μας δεν ελέγχουμε αν είμαστε μέσα στα όρια του πίνακα. Δεδομένου ότι ο πίνακας `p` είναι απροσδιόριστου μεγέθους, πώς θα αντιμετωπίζατε αυτό το πρόβλημα; Αυτό ήταν αρκετό για να λειτουργήσει σωστά το πρόγραμμα. Αν φταίει και κάτι άλλο, προσπαθήστε να το βρείτε με τη χρήση του gdb.



## ΕΡΓΑΣΤΗΡΙΟ 9: Συμβολοσειρές και Ορίσματα Γραμμής Εντολής

Στο εργαστήριο αυτό θα δούμε πώς ορίζονται και πώς χρησιμοποιούνται οι συμβολοσειρές στην C. Επίσης, θα μελετήσουμε κάποιες από τις συναρτήσεις της πρότυπης βιβλιοθήκης της C, που διευκολύνουν την επεξεργασία των συμβολοσειρών, τα πρωτότυπα των οποίων ορίζονται στο αρχείο επικεφαλίδας `string.h`. Τέλος, θα δούμε πώς να διαχειριζόμαστε στο πρόγραμμά μας τα ορίσματα που δίνονται στη γραμμή εντολής κατά την εκτέλεση ενός προγράμματος.

### Άσκηση 1: Επεξεργασία συμβολοσειρών

**1.1** Υλοποιήστε τη συνάρτηση `int mystrlen(char *s)` η οποία δέχεται σαν όρισμα μία συμβολοσειρά και επιστρέφει το μήκος της (χωρίς να συμπεριλαμβάνεται ο χαρακτήρας τέλους συμβολοσειράς `'\0'`).

**1.2** Υλοποιήστε τη συνάρτηση `char *mystrcat(char *s1, char *s2)` η οποία προσαρτά ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

### Άσκηση 2: Χρήση συναρτήσεων που δηλώνονται στο `string.h`

**2.1** Κατασκευάστε το πρόγραμμα `string.c`, το οποίο θα συμπεριλαμβάνει το αρχείο επικεφαλίδας `string.h` και θα πραγματοποιεί το σενάριο που ακολουθεί. Επίσης, ενσωματώστε στο πρόγραμμα και τις συναρτήσεις που υλοποιήσατε στην προηγούμενη άσκηση.

**2.1.1** Ορίστε τις συμβολοσειρές `strA` και `strB` με στατική ή δυναμική δέσμευση μνήμης 80 χαρακτήρων.

**2.1.2** Αντιγράψτε στην `strA` τη συμβολοσειρά `"This is a string."` και στην `strB` τη συμβολοσειρά `"This is another string."`.

```
char *strcpy(char *s1, const char *s2)
```

Αντιγράφει τη συμβολοσειρά `s2` στην `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

**2.1.3** Εκτυπώστε τις δύο συμβολοσειρές και το μήκος τους. Υπολογίστε το μήκος της `strA` μέσω της συνάρτησης `mystrlen` που υλοποιήσατε στην άσκηση 1 και το μήκος της `strB` μέσω της συνάρτησης `strlen` της C.

```
int strlen(const char *s)
```

Επιστρέφει στο όνομα της το μήκος της συμβολοσειράς `s` (χωρίς να μετράται το τελικό `'\0'`).

**2.1.4** Συγκρίνετε αλφαβητικά τις συμβολοσειρές `strA` και `strB` εκτυπώνοντας κατάλληλο μήνυμα.

```
int strcmp(const char *s1, const char *s2)
```

Συγκρίνει τις συμβολοσειρές `s1` και `s2` χαρακτήρα προς χαρακτήρα με βάση τους αντίστοιχους ASCII κωδικούς. Επιστρέφει:

- = 0, αν οι συμβολοσειρές είναι ίδιες
- > 0, αν η `s1` είναι λεξικογραφικά «μεγαλύτερη» της `s2`
- < 0, αν η `s1` είναι λεξικογραφικά «μικρότερη» της `s2`

**2.1.5** Προσαρτήστε τη συμβολοσειρά `strB` στο τέλος της `strA` (χρησιμοποιώντας τη συνάρτηση `mystrcat` που υλοποιήσατε στην άσκηση 1) και εκτυπώστε το αποτέλεσμα της προσάρτησης. Στη συνέχεια, προσαρτήστε τη νέα τιμή της συμβολοσειράς `strA` στο τέλος της `strB` (χρησιμοποιώντας τη συνάρτηση `strcat` της C) και εκτυπώστε το αποτέλεσμα της προσάρτησης.

```
char *strcat(char *s1, const char *s2)
```

Προσαρτά ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

**2.1.6** Χρησιμοποιήστε τη συνάρτηση `strtok` για να εκτυπώσετε μία προς μία τις λέξεις που εμφανίζονται στην τελική συμβολοσειρά `strB`, χωρίς τους χαρακτήρες στίξης.

```
char *strtok(char *string, const char *delim)
```

Αν το `string` δεν είναι `NULL`, η `strtok` ψάχνει στο `string` για την πρώτη εμφάνιση συμβολοσειράς που περιορίζεται από έναν από τους χαρακτήρες που εμφανίζονται στη συμβολοσειρά `delim`. Αν υπάρχει, αντικαθιστά τον χαρακτήρα που βρέθηκε στο `string`, με `'\0'` και επιστρέφει έναν δείκτη στην αρχή του `string`.

Σε κάθε επόμενη κλήση της, η `strtok` καλείται με `NULL` στο πρώτο όρισμα και συνεχίζει τη λειτουργία της από το σημείο που η τελευταία κλήση βρήκε τον χαρακτήρα διαχωρισμού.

Παράδειγμα χρήσης:

```
char *p, s[] = "Little by little, one travels far.";
p = strtok(s, " ,.");
while(p != NULL)
{
    printf("%s\n", p);
    p = strtok(NULL, " ,.");
}
```

Έξοδος προγράμματος:

```
Little
by
little
one
travels
far
```

**Άσκηση 3:** Ορίσματα γραμμής εντολής

**3.1** Κατασκευάστε το πρόγραμμα `calc.c` που να εκτελεί απλές αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμό, πηλίκο διαίρεσης και υπόλοιπο διαίρεσης) μεταξύ ακεραίων. Οι πράξεις που θα γίνονται να δίνονται σαν ορίσματα στη γραμμή εντολής.

Παραδείγματα εκτέλεσης:

```
% ./calc 12 + 18
30
% ./calc 70 % 12
10
```

**Ορίσματα Γραμμής Εντολής**

Ορισμός της συνάρτησης `main`:

```
int main(int argc, char *argv[])
```

- Η μεταβλητή `argc` αρχικοποιείται με το πλήθος των ορισμάτων ( $N$ ) + 1, τα οποία βρίσκονται στις θέσεις `argv[1], ..., argv[N]` του πίνακα συμβολοσειρών `argv`. Το `argv[0]` είναι το όνομα του προγράμματος και το `argv[N+1]` είναι `NULL`.

- Η συνάρτηση `int atoi(const char *s)` επιστρέφει στο όνομά της την αριθμητική τιμή που αντιστοιχεί στην (αριθμητική) συμβολοσειρά `s`.

**ΠΑΡΑΡΤΗΜΑ:** Αποσφαλμάτωση προγραμμάτων (Πράξη 4<sup>η</sup>)

Στο εργαστήριο 8 είδαμε ένα εργαλείο για την αποσφαλμάτωση προγραμμάτων, τον debugger gdb. Στο σημερινό εργαστήριο, θα δούμε πώς μπορούμε να χρησιμοποιήσουμε τον debugger του Dev-C++ για να εντοπίσουμε και να διορθώσουμε σφάλματα διαχείρισης μνήμης και, γενικότερα, λογικά σφάλματα που υπάρχουν στα προγράμματά μας.

Χρήσιμοι σύνδεσμοι:

<http://sourceware.org/gdb/current/onlinedocs/gdb/>

<http://cgi.di.uoa.gr/~ip/debug.html>

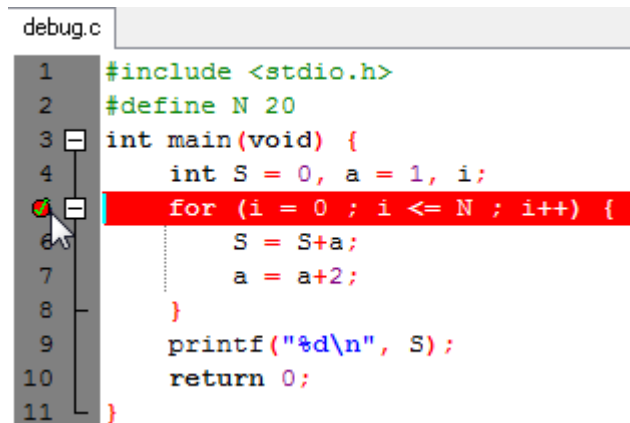
**Ο debugger του Dev-C++**

Έστω ότι θέλουμε να υπολογίσουμε το άθροισμα  $1+3+5+\dots$  για τους πρώτους 20 όρους. Το παρακάτω πρόγραμμα προσπαθεί να αντιμετωπίσει αυτό το πρόβλημα.

```
#include <stdio.h>
#define N 20

int main(void) {
    int S = 0, a = 1, i;
    for (i = 0 ; i <= N ; i++) {
        S = S+a;
        a = a+2;
    }
    printf("%d\n", S);
    return 0;
}
```

Το πρόγραμμα αυτό εκτυπώνει 441 και όχι 400 όπως θα ήταν η σωστή απάντηση. Η διαδικασία που ακολουθούμε για να εντοπίσουμε το λάθος είναι η εξής. Πρώτα, μεταγλωττίζουμε το πρόγραμμά μας. Μετά, βάζουμε ένα breakpoint σε κάποια γραμμή του κώδικα που μας ενδιαφέρει.



The screenshot shows a code editor window titled 'debug.c'. The code is as follows:

```
1 #include <stdio.h>
2 #define N 20
3 int main(void) {
4     int S = 0, a = 1, i;
5     for (i = 0 ; i <= N ; i++) {
6         S = S+a;
7         a = a+2;
8     }
9     printf("%d\n", S);
10    return 0;
11 }
```

A red highlight is placed over line 5, and a breakpoint icon (a small circle with a vertical line) is positioned to the left of the line number 5 in the editor's margin.

Αυτό το καταφέρνουμε κάνοντας κλικ δίπλα από τη γραμμή που μας ενδιαφέρει. Ένα breakpoint είναι κάποιο σημείο στο κώδικα που όταν ο έλεγχος φτάσει σε αυτό, θα σταματήσει η εκτέλεση και θα περιμένει κάποια περαιτέρω δικιά μας ενέργεια. Όπως είναι λογικό, breakpoints βάζουμε σε γραμμές του προγράμματος που είναι κώδικας και όχι δηλώσεις, σχόλια κλπ. Μπορούμε να βάλουμε όσα breakpoints θέλουμε, και τα αφαιρούμε κάνοντας πάλι κλικ δίπλα από τη γραμμή του κώδικα.

Στη συνέχεια, πατάμε το κουμπί debug και ξεκινάμε την διαδικασία της αποσφαλμάτωσης μέσα από το περιβάλλον του Dev-C++. Ο έλεγχος φτάνει στο breakpoint που θέσαμε πριν και έχουμε πια τις εξής επιλογές:



**Debug:** Ξεκινάει τη διαδικασία της αποσφαλμάτωσης.

**Stop Execution:** Αν εκτελείται ήδη αποσφαλμάτωση, τερματίζει τη διαδικασία.

**Add Watch (display):** Δίνοντας το όνομα κάποιας μεταβλητής, μας εμφανίζει συνέχεια στα δεξιά του παραθύρου τι τιμές παίρνει αυτή η μεταβλητή. Το ίδιο μπορούμε να καταφέρουμε αν αφήσουμε για λίγο τον κέρσορα πάνω από τη μεταβλητή αυτή.

**View CPU window:** Παρατηρούμε την κατάσταση του επεξεργαστή σε μια προχωρημένη προβολή (καταχωρητές και κώδικα assembly που εκτελείται), αλλά βλέπουμε και το χρήσιμο υποπαράθυρο “backtrace”, όπου περιέχεται η στοιβά εκτέλεσης και έτσι ξέρουμε ποιες συναρτήσεις δεν έχουν τερματίσει ακόμα.

**Next Line (next):** Συνεχίζει η εκτέλεση του προγράμματος κατά μία γραμμή, χωρίς να μπούμε στο σώμα συναρτήσεων.

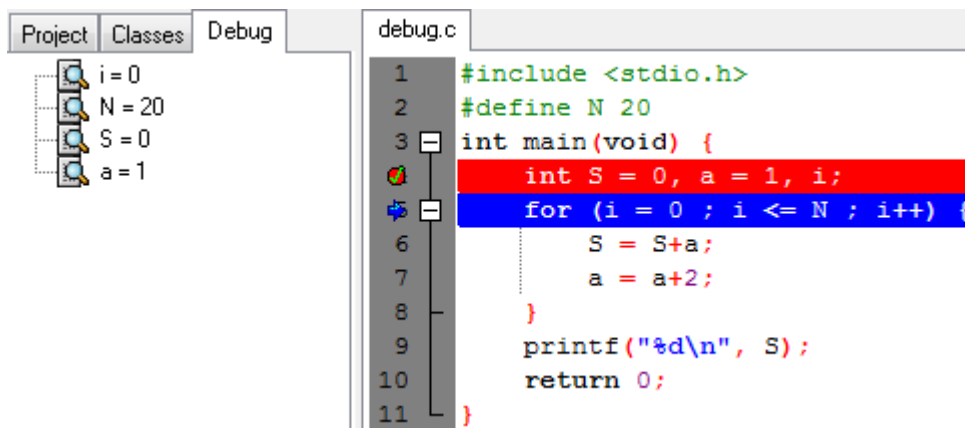
**Into function (step):** Συνεχίζει η εκτέλεση του προγράμματος μέχρι τη πρώτη εντολή της επόμενης γραμμής (π.χ. χρήσιμο ώστε να μη δούμε ολόκληρη την εκτέλεση μιας switch)

**Continue (continue):** Συνεχίζει η εκτέλεση του προγράμματος κανονικά μέχρι το επόμενο breakpoint ή, αν δεν υπάρχει κάποιο άλλο, μέχρι το τέλος του προγράμματος.

**Skip function (finish):** Συνεχίζει η εκτέλεση του προγράμματος κανονικά τουλάχιστον μέχρι το τερματισμό της τωρινής συνάρτησης και το επόμενο breakpoint.

**Next Instruction (nexti):** Εκτελείται η αμέσως επόμενη εντολή. Αν αυτή είναι κλήση συνάρτησης, γίνεται ενιαία εκτέλεση όλου του σώματος της συνάρτησης.

**Into Instruction (stepi):** Αν η αμέσως επόμενη εντολή είναι κλήση συνάρτησης, αντί να εκτελεστεί ενιαία, “μπαίνουμε” μέσα στο σώμα της συνάρτησης και μπορούμε στη συνέχεια να εκτελέσουμε μία-μία τις εντολές της. Αν δεν έχουμε τον πηγαίο κώδικα για την συνάρτηση (για παράδειγμα για την printf), δεν θα μπορούσαμε να μπούμε μέσα στο σώμα της.



Με τη βοήθεια όλων αυτών, μπορούμε να διαπιστώσουμε ότι μετά το τέλος της επανάληψης, το  $i$  έχει την τιμή 21, άρα έγινε μια παραπάνω επανάληψη από όσες θέλαμε (το  $i$  παίρνει τιμές αρχίζοντας από το 0). Άρα η λύση στο πρόβλημα μας είναι να αλλάξουμε αυτή τη γραμμή κώδικα

```
for (i=0 ; i<=N ; i++) {
```

σε αυτή

```
for (i=0 ; i<N ; i++) {
```

## Συμβουλές

Όποτε δουλεύετε με δείκτες στα προγράμματά σας, να έχετε πάντα στο νου σας τα εξής

1. Ένας δείκτης δεν αρχικοποιείται σε `NULL`, αλλά δείχνει σε κάποια τυχαία θέση μνήμης. Είναι πολύ σημαντικό όποτε δηλώνουμε ένα δείκτη να του δίνουμε τιμή `NULL`, έτσι ώστε όποτε χρησιμοποιείται, να ελέγχουμε πρώτα αν η τιμή του είναι `NULL`.
2. Κάθε συμβολοσειρά `char *` πρέπει να έχει αρκετό χώρο για να χωρέσει όλους τους χαρακτήρες που θέλουμε να βάλουμε συν το τελικό `\0`.
3. Όποτε χρησιμοποιούμε πίνακες, πάντα προσέχουμε να μη βγούμε έξω από τα όριά τους.

## Άσκηση

Το παρακάτω πρόγραμμα δημιουργεί έναν πίνακα με τυχαίους μισθούς και βρίσκει το μέσο μισθό, παραλείποντας όσα κελιά έχουν τιμή μικρότερη ή ίση του μηδενός. Όμως το πρόγραμμα δίνει `segmentation fault`, ενώ έχει και λογικά λάθη. Αποσφαλματώστε το με τη βοήθεια ενός debugger.

```
#include <stdio.h>
#define N 100

int main(void) {
    int i = 0, sum = 0;
    int *wages = NULL;
    srand(N);
    for (i = 0 ; i < N ; i++)
        wages[i] = 700 + (rand()%2301) - 1000;
    while (wages[i] > 0 && i < N) {
        sum += wages[i];
        i++; }
    printf("Average wage is %0.2f\n", sum/N);
    return 0;
}
```



## ΕΡΓΑΣΤΗΡΙΟ 10: Δομές και Αυτοαναφορικές Δομές

Στο εργαστήριο αυτό θα μελετήσουμε τη δυνατότητα που μας δίνει η C για να ομαδοποιούμε δεδομένα ορίζοντας δομές. Μέσω των δομών θα ορίσουμε συνδεδεμένες λίστες και δυαδικά δένδρα, που είναι ειδικές δομές οι οποίες ονομάζονται αυτοαναφορικές.

### Άσκηση 1: Δομές και συναρτήσεις

**1.1** Κατασκευάστε το αρχείο `point.c` και ορίστε σε αυτό τη δομή `point` που αποθηκεύει τις συντεταγμένες (τύπου `double`) ενός σημείου στο διδιάστατο χώρο.

**1.2** Ορίστε τη συνάρτηση `struct point middle(struct point a, struct point b)` που υπολογίζει και επιστρέφει το σημείο που είναι το μέσο του ευθυγράμμου τμήματος με άκρα τα σημεία `a` και `b`.

**1.3** Υπολογίστε το μέσο του ευθυγράμμου τμήματος με άκρα τα σημεία  $(1.2, 5.4)$  και  $(7.3, 1.8)$ .

### Άσκηση 2: Δομές και δείκτες

**2.1** Κατασκευάστε το αρχείο `person.c` και ορίστε σε αυτό τη δομή `person` που αποθηκεύει το όνομα, το επώνυμο και το πατρώνυμο ενός ατόμου ως εξής:

```
struct person {
    char *fname;
    char *lname;
    char *mname;
};
```

**2.2** Κατασκευάστε τη συνάρτηση `struct person *person_init(char *firstname, char *lastname, char *middlename)` η οποία δέχεται σαν όρισμα 3 συμβολοσειρές, δεσμεύει χώρο για μία δομή τύπου `person`, την αρχικοποιεί κατάλληλα και επιστρέφει τη διεύθυνση της δομής αυτής στο όνομά της. Έπειτα καλέστε την από τη συνάρτηση `main` με ορίσματα που απεικονίζουν τον πατέρα σας.

**2.3** Ορίστε τη συνάρτηση `struct person *childof(struct person father, char *newname)`, έτσι ώστε να αρχικοποιεί τα στοιχεία ενός παιδιού του `father`, που έχει μικρό όνομα το `newname`, και να επιστρέφει στο όνομά της τη διεύθυνση μίας δομής `person` για το παιδί. Έπειτα, καλέστε την από τη συνάρτηση `main` για να κατασκευάσετε τον εαυτό σας.

### Άσκηση 3: Συνδεδεμένες λίστες

**3.1** Κατασκευάστε το πρόγραμμα `grades.c` στο οποίο να ορίζεται μία αυτοαναφορική δομή λίστας ακεραίων αριθμών.

#### Ορισμός Συνδεδεμένης Λίστας

```
typedef struct listnode *Listptr;  
  
struct listnode {  
    TΔ data;  
    Listptr next;  
};
```

**3.2** Κατασκευάστε τη συνάρτηση `void insert_at_start(Listptr *ptr, int grade)` που να προσθέτει έναν βαθμό στην αρχή της λίστας. Τροποποιήστε την `main` του προγράμματος σας, ώστε να διαβάζει βαθμούς από το πληκτρολόγιο, μέχρι το τέλος της εισόδου, και να τους προσθέτει στην αρχή της λίστας.

**3.3** Κατασκευάστε τη συνάρτηση `float average(Listptr ptr)` που δέχεται σαν όρισμα μία συνδεδεμένη λίστα, διασχίζει τα περιεχόμενα της και υπολογίζει τον μέσο όρο των βαθμών που έχουν αποθηκευθεί σε αυτήν.

## Άσκηση 4: Δυαδικά δένδρα

**4.1** Κατασκευάστε το πρόγραμμα `tree.c` στο οποίο να ορίζεται μία αυτοαναφορική δομή δυαδικού δένδρου ακεραίων αριθμών.

```
typedef struct tnode *Treenptr;

struct tnode {
    TΔ data;
    Treenptr left;
    Treenptr right;
};
```

**4.2** Ένα ταξινομημένο δυαδικό δένδρο είναι ένα δυαδικό δένδρο στο οποίο κάθε κόμβος έχει στο αριστερό του υποδένδρο αριθμούς μικρότερους από τον ίδιο και στο δεξί του υποδένδρο αριθμούς μεγαλύτερους από τον ίδιο, και η ιδιότητα αυτή ισχύει τόσο για το αριστερό όσο και για το δεξί υποδένδρο.

Ορίστε την αναδρομική συνάρτηση `Treenptr addtree(Treenptr p, int x)` η οποία να προσθέτει έναν αριθμό  $x$  στο ταξινομημένο δυαδικό δένδρο  $p$ , διατηρώντας το ταξινομημένο, και να επιστρέφει στο όνομά της το νέο δένδρο. Δηλαδή:

- Αν ο αριθμός που περιέχει ο κόμβος στον οποίο δείχνει ο  $p$  είναι μεγαλύτερος του  $x$ , γίνεται κλήση της `addtree` για το αριστερό παιδί του  $p$ .
- Αν ο αριθμός που περιέχει ο κόμβος στον οποίο δείχνει ο  $p$  είναι μικρότερος του  $x$ , γίνεται κλήση της `addtree` για το δεξί παιδί του  $p$ .
- Αν ο αριθμός που περιέχει ο κόμβος στον οποίο δείχνει ο  $p$  είναι ίσος με το  $x$ , τότε δεν γίνεται καμία εισαγωγή στο δέντρο.
- Αν ο  $p$  είναι `NULL`, κατασκευάζεται ένας νέος κόμβος δένδρου, που περιέχει τον  $x$  και ο  $p$  τίθεται να δείχνει σε αυτόν τον νέο κόμβο.

Τροποποιήστε την `main` του προγράμματός σας, ώστε να διαβάζει αριθμούς από το πληκτρολόγιο, μέχρι το τέλος της εισόδου, και να τους προσθέτει στο δένδρο.

**4.3** Κατασκευάστε την αναδρομική συνάρτηση `void treeprint(Treenptr p)` που δέχεται σαν όρισμα ένα δυαδικό δένδρο και διασχίζει τα περιεχόμενα του σύμφωνα με την εξής λογική:

1. Αν το δέντρο είναι `NULL`, η συνάρτηση επιστρέφει.
2. Αν το δέντρο δεν είναι `NULL`:
  - i. Καλείται η `treeprint` για να εκτυπωθεί το αριστερό παιδί του  $p$ .
  - ii. Εκτυπώνεται η ρίζα του δέντρου.
  - iii. Καλείται η `treeprint` για να εκτυπωθεί το δεξί παιδί του  $p$ .

Καλέστε την `treeprint`, από την `main`, για το δυαδικό δένδρο που έχετε κατασκευάσει και παρατηρήστε την εκτύπωσή του.



## ΕΡΓΑΣΤΗΡΙΟ 11: Είσοδος/Εξόδος - Αρχεία

Στο εργαστήριο αυτό θα μελετήσουμε τους μηχανισμούς εισόδου/εξόδου που μας παρέχει η C. Θα αναφερθούμε στις μονάδες εισόδου/εξόδου, που είναι τα ρεύματα, θα κάνουμε μία επισκόπηση στα προκαθορισμένα ρεύματα και θα ορίσουμε δικά μας ρεύματα για την επεξεργασία αρχείων κειμένου και δυαδικών αρχείων.

### Άσκηση 1: Αρχεία κειμένου

Κατασκευάστε το πρόγραμμα `display.c` που δέχεται ως όρισμα γραμμής εντολής το όνομα ενός αρχείου κειμένου και προβάλλει ανά 20 τις γραμμές του αρχείου, προτρέποντας τον χρήστη να συνεχίσει, αν επιθυμεί, έως ότου συναντήσει το τέλος του αρχείου.

<pre>FILE *fopen(const char *filename,             const char *mode)</pre> <p>Ανοίγει το αρχείο με όνομα <code>filename</code> με τρόπο προσπέλασης που καθορίζεται από το <code>mode</code>:</p> <ul style="list-style-type: none"><li>◦ <code>"r"</code>: Διάβασμα από υπάρχον αρχείο.</li><li>◦ <code>"w"</code>: Γράψιμο σε αρχείο. Αν το αρχείο δεν υπάρχει, δημιουργείται. Αν υπάρχει, διαγράφονται τα περιεχόμενά του και το γράψιμο αρχίζει από την αρχή.</li><li>◦ <code>"a"</code>: Προσάρτηση δεδομένων στο τέλος του αρχείου, χωρίς διαγραφή των υπαρχόντων περιεχομένων του.</li><li>◦ <code>"r+"</code>: Διάβασμα και γράψιμο οπουδήποτε στο αρχείο χωρίς διαγραφή των υπαρχόντων περιεχομένων του.</li><li>◦ <code>"w+"</code>: Διάβασμα και γράψιμο οπουδήποτε στο αρχείο με διαγραφή των υπαρχόντων περιεχομένων του.</li><li>◦ <code>"a+"</code>: Διάβασμα οπουδήποτε μέσα από το αρχείο και γράψιμο μόνο στο τέλος του, χωρίς διαγραφή των υπαρχόντων περιεχομένων του.</li><li>◦ Στις παραπάνω συμβολοσειρές που δείχνουν τον τρόπο προσπέλασης του αρχείου προστίθεται και ο χαρακτήρας <code>b</code>, αν πρόκειται για δυαδικό αρχείο (<code>"rb"</code>, <code>"wb"</code>, <code>"ab"</code>, <code>"rb+"</code> ή <code>"r+b"</code>, <code>"wb+"</code> ή <code>"w+b"</code>, <code>"ab+"</code> ή <code>"a+b"</code>).</li></ul> <p>Επιστρέφει ένα ρεύμα μέσω του οποίου μπορούμε στη συνέχεια να αναφερόμαστε στο αρχείο, ή <code>NULL</code>, αν για κάποιο λόγο δεν ήταν δυνατόν να ανοίξει το αρχείο.</p>	<pre>int fclose(FILE *fp)</pre> <p>Κλείνει το ρεύμα <code>fp</code>. Επιστρέφει 0 σε περίπτωση επιτυχίας.</p> <pre>int feof(FILE *fp)</pre> <p>Επιστρέφει τιμή διάφορη από το 0, αν προηγούμενο διάβασμα απέτυχε λόγω του ότι τα δεδομένα στο αρχείο έχουν τελειώσει, αλλιώς επιστρέφει 0.</p> <pre>int fprintf(FILE *fp, ...) int fscanf(FILE *fp, ...)</pre> <p>Ίδιες με τις <code>printf</code> και <code>scanf</code>, μόνο που γράφουν στο ρεύμα <code>fp</code> αντί του <code>stdout</code> ή διαβάζουν από το ρεύμα <code>fp</code> αντί του <code>stdin</code>, αντίστοιχα.</p> <pre>char *fgets(char *buf, int max,             FILE *fp)</pre> <p>Διαβάζει το πολύ <code>max-1</code> χαρακτήρες από το ρεύμα <code>fp</code> μέχρι την αλλαγή γραμμής και τους φυλάσσει στο <code>buf</code> (συμπεριλαμβανομένης και της αλλαγής γραμμής). Αν δεν υπάρχουν δεδομένα για διάβασμα από το ρεύμα, επιστρέφει <code>NULL</code>.</p> <pre>int getc(FILE *fp)</pre> <p>Επιστρέφει τον επόμενο χαρακτήρα από το ρεύμα <code>fp</code>, η <code>EOF</code>, αν έχουμε φτάσει στο τέλος του αρχείου.</p>
---	---

## Άσκηση 2: Δυαδικά αρχεία

**2.1** Κατασκευάστε το πρόγραμμα `grades.c`, το οποίο να ανοίγει το αρχείο `grades.dat` για γράψιμο σε δυαδική μορφή. Στη συνέχεια, να διαβάζει ένα όνομα (συμβολοσειρά) από την πρότυπη είσοδο και έναν βαθμό και να τα γράφει στο αρχείο. Το πρόγραμμα να τερματίζει όταν επισημανθεί, με κάποιο τρόπο, το τέλος της εισόδου.

**2.2** Στη συνέχεια, επεκτείνετε το πρόγραμμά σας, ώστε να ανοίγει το αρχείο `grades.dat` για διάβασμα, να διαβάζει τα δεδομένα που έχουν γραφεί σ' αυτό και να τα προβάλλει στην οθόνη.

```
size_t fread(void *ptr, size_t size,
             size_t count, FILE *fp)
```

Διαβάζει από το ρεύμα `fp`, το πολύ `count` δεδομένα μεγέθους `size` το καθένα και τα τοποθετεί από τη διεύθυνση `ptr` και μετά.

```
size_t fwrite(const void *ptr,
             size_t size,
             size_t count,
             FILE *fp)
```

Γράφει στο ρεύμα `fp` το πολύ `count` δεδομένα μεγέθους `size` το καθένα, παίρνοντάς τα από τη διεύθυνση `ptr` και μετά.

Για να δείτε τα περιεχόμενα του αρχείου `grades.dat`, αν δουλεύετε σ' ένα Unix σύστημα, χρησιμοποιήστε την εντολή "`od -tulc grades.dat`".

**Άσκηση 3:** Δημιουργήστε το πρόγραμμα `compare.c`, το οποίο να δέχεται στη γραμμή εντολής τα ονόματα δύο αρχείων και να ελέγχει αν τα αρχεία αυτά είναι ίδια byte προς byte.

**Άσκηση 4:** Στην άσκηση αυτή θα υλοποιήσουμε ένα υποσύνολο της εντολής `wc` του Unix. Συγκεκριμένα, θα κατασκευάσουμε πρόγραμμα που θα μετράει το πλήθος των χαρακτήρων και το πλήθος των γραμμών ενός αρχείου κειμένου.

**4.1** Κατασκευάστε το πρόγραμμα `count.c` που να δέχεται ως όρισμα γραμμής εντολής το όνομα ενός αρχείου κειμένου, να το ανοίγει για διάβασμα, να μετράει το πλήθος των χαρακτήρων του αρχείου και να προβάλλει το αποτέλεσμα στην οθόνη.

**4.2** Τροποποιήστε το πρόγραμμά σας, ώστε να μετράει και το πλήθος των γραμμών του αρχείου.