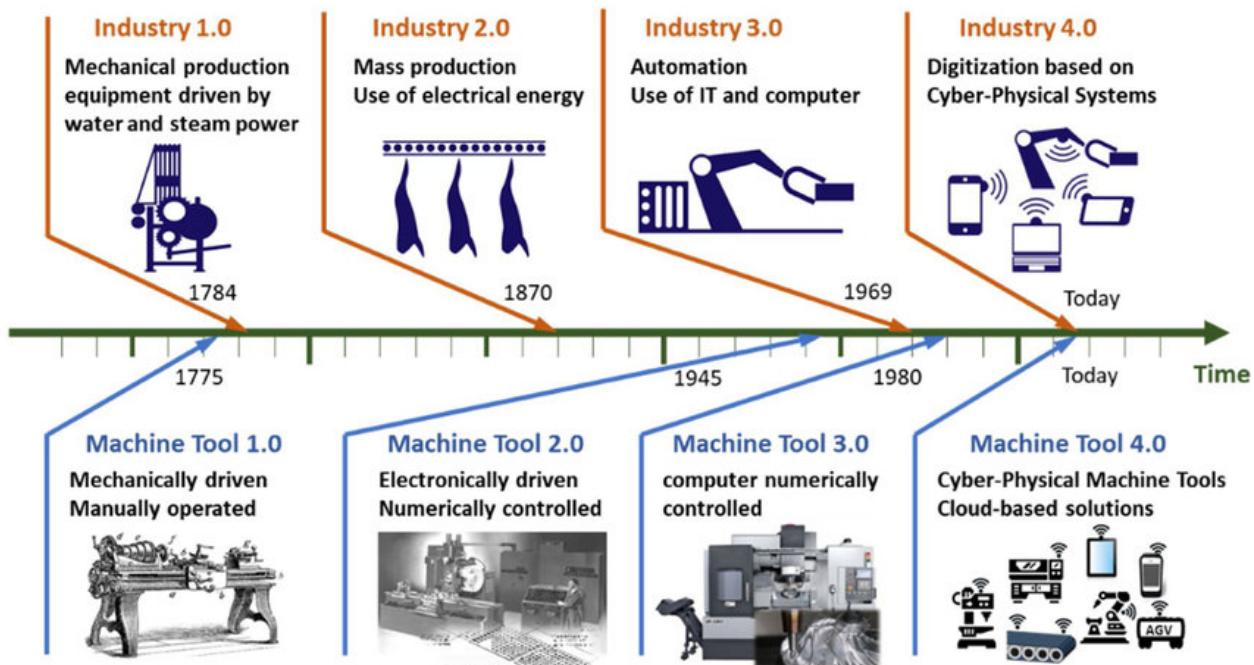


Εργαστήριο #0: Καλημέρα Κόσμε!

Εισαγωγή, Χρήσιμες Εφαρμογές



A man is only as good as his tools. -- Emmert Wolf

Καλωσήρθατε στο εργαστήριο! Σκοπός των πρώτων συναντήσεών μας θα είναι η εξοικείωση με τα βασικά εργαλεία του μαθήματος. Θα προσπαθήσουμε να καλύψουμε τα ακόλουθα:

1. Να διαχειριζόμαστε το ακαδημαϊκό μας email με το Webmail.
2. Να εγγραφούμε στο ηλεκτρονικό φόρουμ του μαθήματος Piazza.
3. Να ενεργοποιήσουμε το λογαριασμό μας στο εργαστήριο Linux.
4. Να συνδεόμαστε απομακρυσμένα στα μηχανήματα της σχολής με ssh.
5. Να εκτελέσουμε τις πρώτες μας εντολές σε Linux shell.
6. Να χρησιμοποιούμε εργαλεία επεξεργασίας κώδικα editor.
7. Να εκτελέσουμε το πρώτο μας Hello World πρόγραμμα!

Το υλικό των εργαστηρίων είναι αποτέλεσμα προσπάθειας πολλών και εξαίρετων συνεργατών. Θερμές ευχαριστίες στους Δημήτρη Ψούνη (ο οποίος έχει και [εξαιρετικό κανάλι με εκπαιδευτικό υλικό](#)), Στέφανο Σταμάτη, Νίκο Ποθητό, Μάνο Καρβούνη, Γιώργο Καστρίνη, Βασίλη Αναστασίου και στον Δρ. Ιωάννη Χαμόδρακα για τη συνεισφορά τους. Το υλικό των φυλλαδίων είναι βασισμένο στο αντίστοιχο υλικό που αναπτύχθηκε για το μάθημα από τον καθηγητή Παναγιώτη Σταματόπουλο τον οποίο ευχαριστούμε θερμότατα.

webmail

If you just communicate, you can get by. But if you communicate skillfully, you can work miracles. -- Jim Rohn

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

Σύνδεση (Login)

Ενώ είμαστε συνδεδεμένοι στο Internet, ανοίγουμε ένα παράθυρο φυλλομετρητή (browser - Firefox, Chrome, Brave ή ακόμα και Lynx αν είστε γενναία παιδιά) και πληκτρολογούμε την διεύθυνση <http://webmail.noc.uoa.gr>, οπότε και εμφανίζεται στην ιστοσελίδα η προτροπή για εισαγωγή των στοιχείων μας.



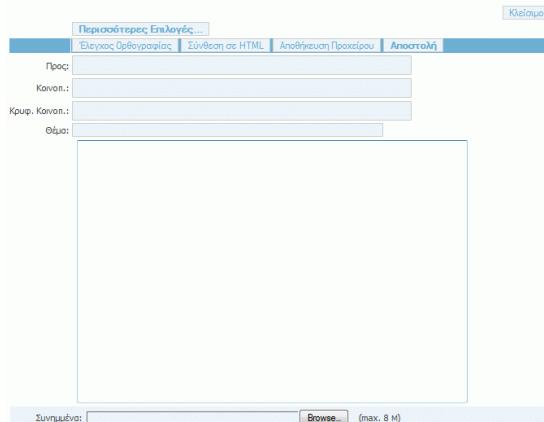
Πληκτρολογούμε το όνομα χρήστη sdiXXYYYYY και τον κωδικό μας και πατάμε το κουμπί «Σύνδεση» (δεν έχετε λογαριασμό ακόμα; ακολουθήστε τις οδηγίες [εδώ](#)).

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 (To): Συμπληρώνουμε την ηλεκτρονική διεύθυνση του αποδέκτη, ή τις ηλεκτρονικές διευθύνσεις χωρισμένες με κόμματα (εφόσον θέλουμε να το αποστείλουμε σε πολλαπλούς αποδέκτες).
- Κοινοπ. (CC): Συμπληρώνουμε τις ηλεκτρονικές διευθύνσεις αυτών στους οποίους κοινοποιείται το μήνυμα.
- Κρυψ. Κοινοπ. (BCC): Συμπληρώνουμε τις ηλεκτρονικές διευθύνσεις αυτών που θέλουμε να λάβουν το μήνυμα χωρίς να εμφανίζονται οι διευθύνσεις τους σε αυτούς που λαμβάνουν το μήνυμα.
- Θέμα (Topic): Συμπληρώνουμε το θέμα του μηνύματος - φροντίζουμε να είναι κάτι περιεκτικό και ευανάγνωστο.
- 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/ry8mieseis.html>.

piazza

Όπως ίσως ακούσατε - αν παρευρεθήκατε στην 1η διάλεξη - στο μάθημα υπάρχει ηλεκτρονικό φόρουμ συζήτησης, μέσω του οποίου ενημερωνόμαστε για τις ανακοινώσεις του μαθήματος, κάνουμε απορίες ή ανταλλάσσουμε απόψεις γενικά για θέματα προγραμματισμού και ότι άλλο σχετικό με το μάθημα.

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

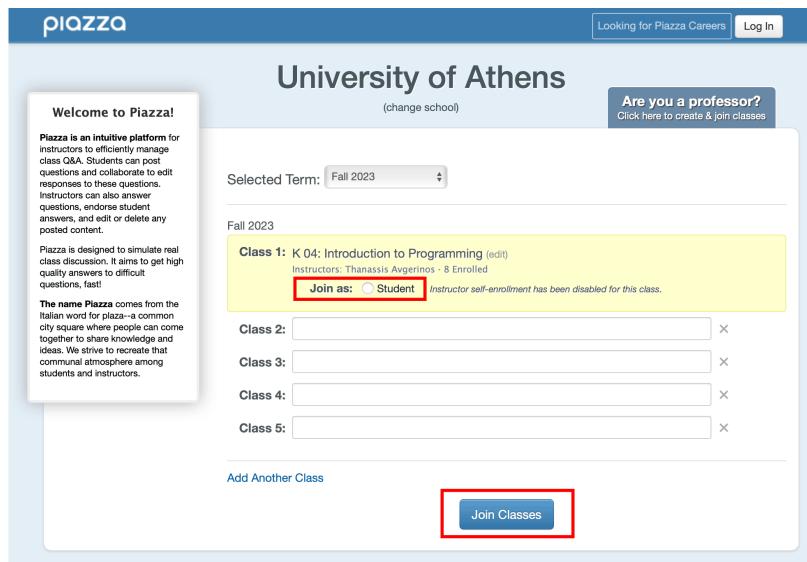
<https://progintro.github.io>

Στην ενότητα «Επικοινωνία» επιλέγουμε το σύνδεσμο για την εγγραφή στο Piazza

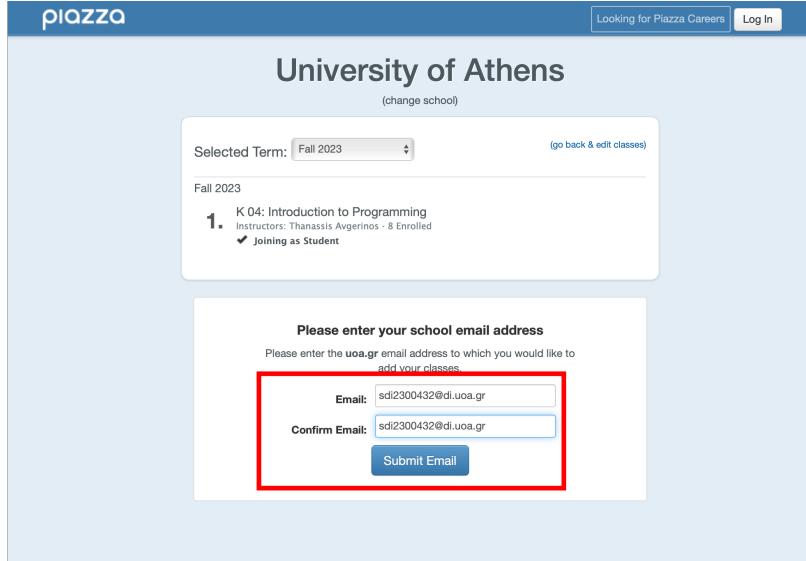
Επικοινωνία

Ερωτήσεις για διαδικαστικά, το μάθημα και τις ασκήσεις αποκλειστικά στο **Piazza** - απαιτεί di.uoa.gr email. Δεν έχεις ακόμα di.uoa.gr email; Βεβαιώσου ότι συμπλήρωσες την φόρμα στην Συμμετοχή παραπάνω και θα σου στείλουμε πρόσκληση.

Στη σελίδα του Piazza επιλέγουμε το "Join as: student" και πατάμε το "Join Classes" στο κάτω μέρος της φόρμας.



Στην επόμενη σελίδα συμπληρώνουμε το ακαδημαϊκό μας email και πατάμε "Submit Email".



Από το webmail ανοίγουμε το email που θα έχει έρθει από το Piazza και αντιγράφουμε το activation code που υπάρχει σε αυτό.

Θέμα: Welcome to Piazza! You've been enrolled in K 04 as a student.
 Από: "Piazza" <no-reply@piazza.com>
 Ημερομηνία: Κυριακή 1, Οκτώβριος 2023 8:59
 Προς: sdi2300432@di.cka.gr
[Περισσότερες Επιλογές...](#)

Welcome to Piazza!
 sdi2300432@di.cka.gr was just enrolled as a student in **K 04** on Piazza, the social site for class Q&A!

Get Started.
 Follow this link to activate your account and set your password: https://www.piazza.com/first_login?token=scc9kKMfkob
 Or, if prompted, enter this activation code: **scc9kKMfkob**

Already Have an Account?
 If you already have a Piazza account under a different email address, link stef@di.cka.gr to your account [here](#).

What is Piazza?
 Piazza is an online gathering place where students can ask, answer, and explore 24/7 under the guidance of their instructors. Thousands of classes are using Piazza, and the fastest has an average response time of 7 minutes! We notify your class when you're stuck, let you ask and answer questions anonymously, and support LaTeX and code formatting. There are even free iPhone and Android apps.

**Please be sure to add "no-reply@piazza.com" to your contacts list to prevent our emails from getting stuck in your spam folder.
 **You can configure your email preferences by visiting your Account Settings page.

Thanks,
 The Piazza Team
 --
 Contact us at team@piazza.com

Συμπληρώνουμε το activation code στη σελίδα του Piazza και πατάμε το "Submit Code".

Στην επόμενη σελίδα συμπληρώνουμε το πλήρες ονοματεπώνυμο μας, δύο φορές τον κωδικό πρόσβασης που θα έχουμε για να συνδεόμαστε στο Piazza, κάποια στοιχεία για τις σπουδές μας, επιλέγουμε τη συμφωνία με τους όρους χρήσης της υπηρεσίας και τέλος πατάμε το "continue".

Έπειτα επιλέγουμε το "Don't join the network".

In the network

- ✓ Access class Q&A discussions
- ✓ See what classes other students are taking
- ✓ See where your classmates have worked
- ✓ See examples of strong resumes in your community
- ✓ Get referred to companies by your classmates
- ✓ Know when companies are coming to campus
- ✓ Get contacted by companies for employment opportunities

Outside the network

- ✓
- ✗
- ✓
- ✗
- ✓
- ✗
- ✓
- ✗
- ✓
- ✗

[Join the network](#) [Don't join the network](#)

Learn more about how Piazza complies with FERPA

Πλέον ο λογαριασμός μας στο Piazza είναι έτοιμος!

Μπορούμε να επιλέξουμε τα μηνύματα από τη λίστα στα αριστερά για να τα δούμε, να απαντήσουμε στο κάτω μέρος σε κάποιο από αυτά, ή να στείλουμε κάποιο νέο από την επιλογή "New Post" που βρίσκεται στο πάνω μέρος της λίστας.

Καλώς ήρθατε στο Piazza!

Καλώς ήρθατε!

Όλες οι συζητήσεις για το μάθημα φέτος θα γίνουν μέσου του Piazza (παρακαλώ μην στέλνετε emails). Προτού κάνετε μια κανονική ερώτηση, ελέγχετε μήπως έχει ήδη απαντηθεί.

Σας ευχαριστώ και καλή αρχή!

Θανάσης Αυγερινός

[good note](#) | 0 Updated 1 day ago by Thanassis Avgerinos

followup discussions, for lingering questions and comments

Start a new followup discussion

Compose a new followup discussion

Average Response... Special Mentions: N/A There are no special mentions at this time. Online Now | ... 1 | 3

Copyright © 2022 Piazza Technologies, Inc. All Rights Reserved. [Privacy Policy](#) [Copyright Policy](#) [Terms of Use](#) [Report Bug](#)

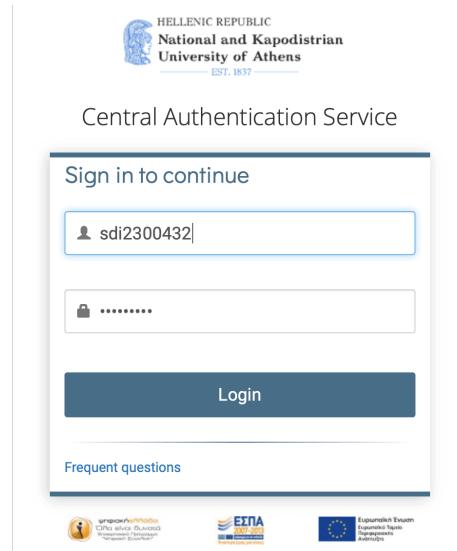
Πως μπορούμε να κάνουμε ερωτήσεις με τρόπο που βελτιστοποιεί την πιθανότητα του να πάρουμε απάντηση; Μπορείτε να βρείτε έναν χρήσιμο οδηγό [εδώ](#).

linux

All the best people in life seem to like LINUX. -- Steve Wozniak

Το Τμήμα παρέχει σε κάθε φοιτητή, λογαριασμό για την πρόσβαση στα μηχανήματα του εργαστηρίου Linux. Για να μπορέσουμε να χρησιμοποιήσουμε το λογαριασμό αυτό, θα πρέπει να τον ενεργοποιήσουμε ορίζοντας έναν κωδικό με τον οποίο θα συνδεόμαστε. Αυτή η διαδικασία μπορεί να γίνει μέσω της ιστοσελίδας <https://account.di.uoa.gr/>

Ανοίγουμε έναν παράθυρο φυλλομετρητή (browser) σε ανώνυμη περιήγηση και πληκτρολογούμε την παραπάνω διεύθυνση. Στη συνέχεια συνδεόμαστε στην υπηρεσία με τον ακαδημαϊκό μας λογαριασμό.



Αφότου συνδεθούμε και μας δείξει τα στοιχεία μας, επιλέγουμε το μπλε κουμπί με το λουκέτο.

Στην σελίδα που θα ανοίξει ορίζουμε τον κωδικό με τον οποίο θα συνδεόμαστε στο λογαριασμό μας για το εργαστήριο Linux. Πρέπει να τον πληκτρολογήσουμε δύο φορές και να πατήσουμε Αποθήκευση. Μπορούμε να επιλέξουμε οτιδήποτε θέλουμε, αρκεί να πληροί όλους τους κανόνες που αναφέρονται πιο κάτω στην ίδια σελίδα.

 Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Υπηρεσία διαχείρισης κωδικού εργαστηρίου

Συνδεθήκατε ως: sdi2300432 Αποσύνδεση

Ανάθεση κωδικού πρόσβασης

Χρήστης: sdi2300432 [Dokimopoulos Christos]

Νέο συνθηματικό:

Επαλήθευση:

Για να γίνει αποδεκτός ο κωδικός πρέπει:

- να αποτελείται αποκλειστικά από λατινικά κεφαλαία ή πεζά γράμματα, αριθμούς και σύμβολα
- να περιέχει τουλάχιστον 3 από τις 4 παραπάνω κατηγορίες χαρακτήρων (κεφαλαία, πεζά, αριθμούς, σύμβολα)
- να περιέχει τουλάχιστον 8 χαρακτήρες
- τουλάχιστον το 60% των χαρακτήρων να είναι διαφορετικού μεταξύ τους
- να μην επαναλαμβάνεται κάποιος χαρακτήρας σε πάνω από το 25% του συνολικού μήκους
- να μην ταυτίζεται με τους 2 προηγουμένους κωδικούς του ίδιου χρήστη

Copyright E.K.P.A. 2021 | All Rights Reserved

Αν ο κωδικός που πληκτρολογήσαμε πληροί όλους τους κανόνες, η εφαρμογή μας ενημερώνει ότι η διαδικασία ολοκληρώθηκε με επιτυχία. Πατάμε Αποσύνδεση και κλείνουμε αυτό το παράθυρο περιήγησης.

 Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Υπηρεσία διαχείρισης κωδικού εργαστηρίου

Συνδεθήκατε ως: sdi2300432 Αποσύνδεση

Ο κωδικός άλλαξε επιτυχώς

Copyright E.K.P.A. 2021 | All Rights Reserved

Αν ο κωδικός που πληκτρολογήσαμε δεν πληρούσε κάποιον από τους κανόνες, θα εμφανιστεί αντίστοιχο μήνυμα σφάλματος και θα πρέπει να δοκιμάσουμε ξανά με διαφορετικό κωδικό που να τους πληροί.

Ο κωδικός πρέπει να περιέχουν τουλάχιστον 8 χαρακτήρες

Μπορούμε να επαναλάβουμε την ίδια διαδικασία και να ορίσουμε νέο κωδικό ανά πάσα στιγμή, είτε σε περίπτωση που ξεχάσουμε, είτε όταν θέλουμε να αλλάξουμε τον κωδικό για το λογαριασμό μας στα μηχανήματα του εργαστηρίου linux του τμήματος.

ssh

Το SSH (Secure Shell) είναι ένα πρωτόκολλο δικτύου που επιτρέπει την ασφαλή απομακρυσμένη σύνδεση και διαχείριση ενός υπολογιστή ή server μέσω κρυπτογραφημένης επικοινωνίας. Χρησιμοποιείται από μηχανικούς λογισμικού για τη διαχείριση servers, τη μεταφορά αρχείων και την εκτέλεση εντολών εξ' αποστάσεως, εξασφαλίζοντας παράλληλα την ασφάλεια των δεδομένων από υποκλοπές και επιθέσεις. Είναι χρήσιμο γιατί επιτρέπει την απομακρυσμένη εργασία και διαχείριση συστημάτων με ασφάλεια, ανεξαρτήτως γεωγραφικής τοποθεσίας. Είχες πάει διακοπές στην Κύθνο και ξαφνικά έσκασε ένα incident στην εφαρμογή που τρέχει σε datacenter στην Καισαριανή; Δεν χρειάζεσαι ούτε πλοίο ούτε αεροπλάνο, μόνο σύνδεση internet. Συνδέσου με SSH στιγμιαία και λύσε όποιο πρόβλημα προκύψει απομακρυσμένα.

Υπάρχουν δύο τρόποι να συνδεθείτε σε ένα από τα συστήματα του εργαστηρίου:

1. Χρησιμοποιώντας την εφαρμογή ssh που έχει το σύστημά σας (native στο Linux, Mac, Windows 11+).
2. Εγκαθιστώντας την εφαρμογή PuTTY και χρησιμοποιώντας την ως την εφαρμογή SSH εάν είστε σε παλαιότερο σύστημα Windows.

Native ssh

Εφόσον το λειτουργικό σας το υποστηρίζει, ο απλούστερος τρόπος να συνδεθείτε σε ένα από τα συστήματα του εργαστηρίου είναι ο εξής:

1. Ανοίγετε ένα τερματικό (terminal ή command prompt).
2. Τρέχετε την ακόλουθη εντολή:

```
ssh sdiXXXXYYYY@linux08.di.uoa.gr
```

3. Αφού αποδεχτείτε το κλειδί και βάλετε τον κωδικό σας μπορείτε να ολοκληρώσετε την σύνδεσή σας:

```
The authenticity of host 'linux08.di.uoa.gr (195.134.65.79)' can't be established.  
ECDSA key fingerprint is SHA256:72FjL3k3ojnkAkjuk5hnysuBZhpnGhpYZPjnLu8nThc.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added 'linux08.di.uoa.gr,195.134.65.79' (ECDSA) to the list of known hosts.  
sdiXXXXYYYY@linux08.di.uoa.gr's password:  
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-186-generic x86_64)
```

- * Documentation: <https://help.ubuntu.com>
- * Management: <https://landscape.canonical.com>
- * Support: <https://ubuntu.com/pro>

Expanded Security Maintenance for Applications is not enabled.

10 updates can be applied immediately.

To see these additional updates run: apt list --upgradable

54 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at <https://ubuntu.com/esm>

*** System restart required ***
linux08:/home/users/sdiXXXXYYYY>

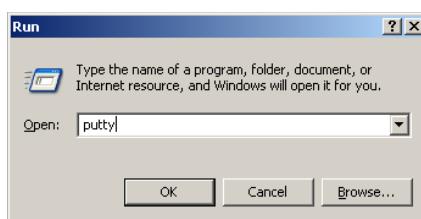
Αντίστοιχα υπάρχουν γύρω στα ~30 υπολογιστικά συστήματα στα οποία μπορείτε να συνδεθείτε με αντίστοιχα domain names (linux01.di.uoa.gr, linux02.di.uoa.gr, ... linux29.di.uoa.gr). Έχουμε 30 συστήματα για δύο λόγους: (1) redundancy - αν ένα σύστημα έχει πρόβλημα, μπορείτε να συνδεθείτε σε κάποιο άλλο, και (2) load balancing - αν μπείτε και οι 300 σε ένα σύστημα ταυτόχρονα το πιθανότερο είναι πως θα καταρρεύσει - αποφύγετε να συνδεθείτε όλοι στο linux01 :). Τα αρχεία σας συγχρονίζονται αυτόματα ανάμεσα σε όλα τα συστήματα.

PuTTY

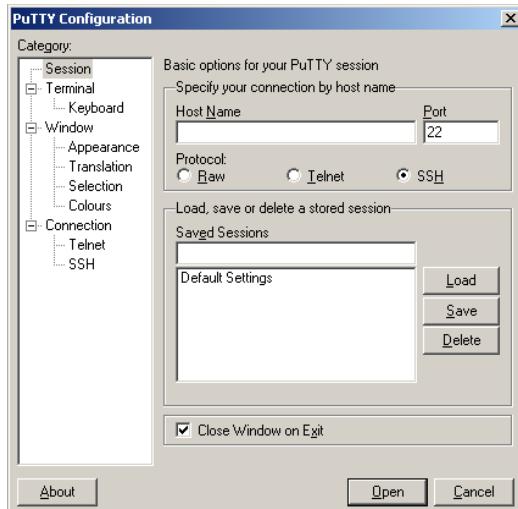
Εάν η πλατφόρμα που χρησιμοποιείτε δεν έχει ssh, θα χρειαστεί να καταφύγετε στο PuTTY. Το PuTTY είναι άλλο ένα πρόγραμμα απομακρυσμένης σύνδεσης, δηλαδή μέσω αυτού μπορούμε να συνδεόμαστε σε απομακρυσμένους υπολογιστές και να δουλεύουμε σαν να καθόμασταν μπροστά σε αυτούς! Έτσι, μπορούμε να συνδεθούμε και να δουλέψουμε στα συστήματα Linux της σχολής.

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>

Πατάμε στα Windows, Start->Run και στο παράθυρο που εμφανίζεται:



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



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 ακολουθόμενο 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 sdiXXYYYYY. 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:



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

shell

Παραπάνω συνδεθήκαμε (με ssh ή PuTTY) σε ένα από τα συστήματα Linux. Το λειτουργικό σύστημα είναι τώρα έτοιμο να αλληλεπιδράσει μαζί μας, περιμένοντας τις εντολές μας για να δράσει αναλόγως.

Η γραμμή στην οποία γράφουμε τις εντολές μας λέγεται command line interface (γραμμή εντολών ή CLI) και το πρόγραμμα που ερμηνεύει αυτές τις εντολές λέγεται shell (sh ή κέλυφος). Σε αντίθεση με τα προγράμματα που απαιτούν γραφικό περιβάλλον (Graphical User Interface - GUI), η γραμμή εντολών απαιτεί χρήση του πληκτρολογίου. Στην αρχή ίσως να σας ξενίσει και να σας κάνει να νιώθετε "δεν ξέρω τι να κάνω σε αυτήν την γραμμή", αλλά σας διαβεβαιώνουμε πως μετά από αρκετή χρήση η γραμμή εντολών γίνεται δεύτερη φύση και ένα πολύ χρήσιμο εργαλείο στην φαρέτρα σας.

Θα μάθουμε διάφορες εντολές κατά την διάρκεια του μαθήματος και θα ξεκινήσουμε με τις δύο που μας επιτρέπουν να βρίσκουμε ονόματα αρχείων στο σύστημά μας: (1) ls και (2) cd.

ls (LiSt)

Ας ξεκινήσουμε! Πρώτα πληκτρολογούμε στην γραμμή εντολών:

ls

και ως αποτέλεσμα βλέπουμε τα περιεχόμενα του καταλόγου 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:

```
linux14:/home/users/thanassis>ls -l
total 676
...
-rw-r--r-- 1 thanassis dep 652 Oct 1 10:25 list.py
-rwxr-xr-x 1 thanassis dep 61258 Nov 23 2023 main*
-rw-r--r-- 1 thanassis dep 400 Nov 22 2023 main.c
drwxr-xr-x 3 thanassis dep 4096 Oct 13 2023 progintro/
drwxr-s--- 6 thanassis www 4096 Oct 3 22:40 public_html/
linux14:/home/users/thanassis>
```

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

- Το πρώτο γράμμα (d ή -) υποδηλώνει αν το αντικείμενο είναι κατάλογος (directory) ή αρχείο (file) 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.

cd (Change Directory)

Για να εισέλθουμε σε έναν κατάλογο (directory) πληκτρολογούμε:

```
cd όνομα_καταλόγου
```

Ας μπούμε τώρα στον κατάλογο `public_html` (ή όποιον άλλο φάκελο φτιάξετε με την εντολή `mkdir`) και να ελέγχουμε τα περιεχόμενα του. Πληκτρολογούμε:

```
cd public_html
```

```
ls -l
```

Για να επιστρέψουμε στον αρχικό κατάλογό 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.

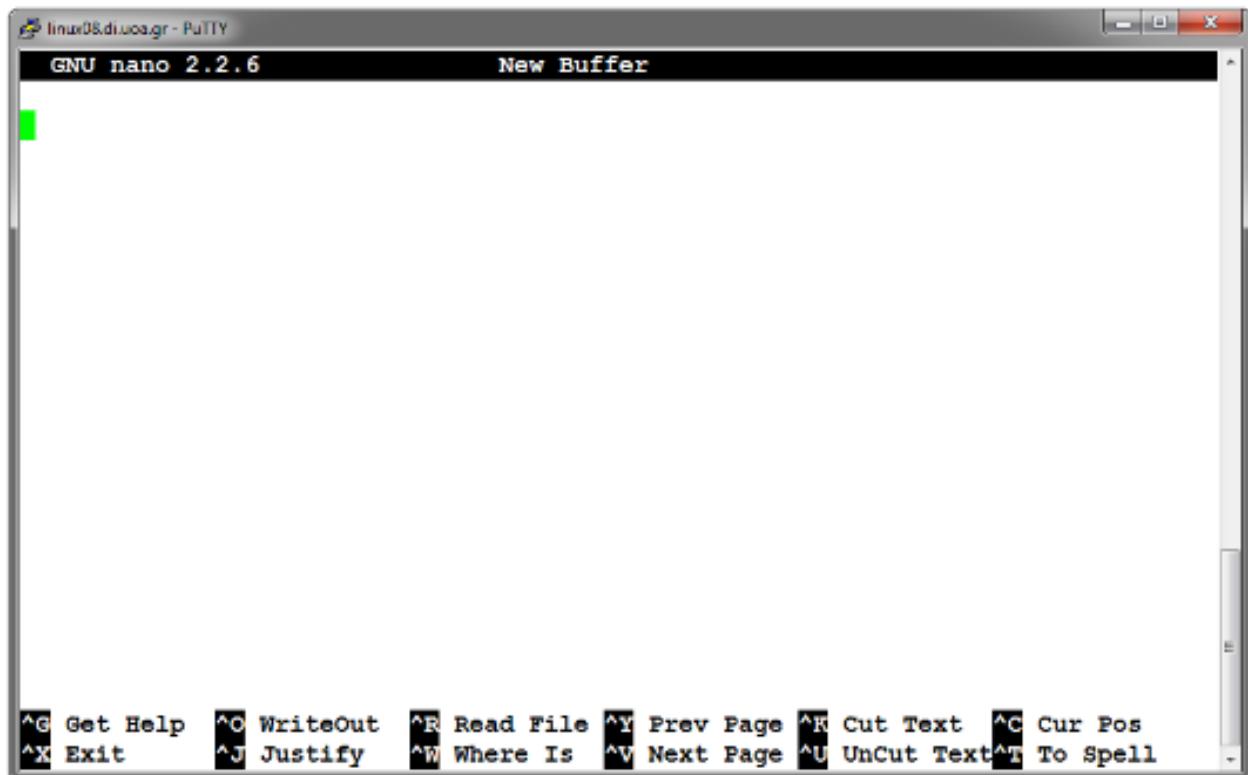
editors

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 (editor) pico. Στα επόμενα μαθήματα θα δούμε και άλλους editors, αλλά εδώ ξεκινάμε από τον πιο μικρό και απλό (εξού και το όνομά του).

Για να ξεκινήσουμε τον 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:

Hotkey	Description
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.
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.

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



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

Για να τυπώσουμε στην οθόνη τα περιεχόμενα του αρχείου που δημιουργήσαμε χρησιμοποιούμε την εντολή cat και πληκτρολογούμε:

```
cat file.txt
```

HelloWorld

Το πρώτο πρόγραμμα που γράφουμε σε κάποια γλώσσα προγραμματισμού, είθισται να είναι το τύπωμα της φράσης "Hello World!". Για να τα καταφέρουμε πρέπει να προσπαθήσουμε να το γράψουμε:

```
/* File: hello.c */
#include <stdio.h>
int main() {
    printf("Hello world!\n");
    return 0;
}
```

Χρησιμοποιώντας τον pico editor, φτιάχνουμε ένα αρχείο hello.c με τα παραπάνω περιεχόμενα. Στην συνέχεια μεταγλωττίζουμε το πρόγραμμά μας:

```
gcc -o hello hello.c
```

Και στην συνέχεια το τρέχουμε:

```
./hello  
Hello world!
```

Αυτό ήταν! Μόλις τρέξατε το πρώτο σας πρόγραμμα!

Άσκηση (από θέμα εξετάσεων)

Γράψτε ένα πρόγραμμα about το οποίο τυπώνει 3 γραμμές στην πρότυπη έξοδο (stdout): 1) το όνομά σας με λατινικούς χαρακτήρες στην 1η, 2) το sdi σας στην 2η και 3) I'm 100% "ready"! στην 3η. Παράδειγμα εξόδου από την εκτέλεση του προγράμματος about:

```
$ ./about  
Michael Jordan  
sdi2300999  
I'm 100% "ready"!  
$
```

Advanced: Τύπωμα emojis / bold / blink

Παραπάνω χρησιμοποιήσαμε την ειδική ακολουθία \n για να τυπώσουμε μια νέα γραμμή μετά το μήνυμά μας. Αντίστοιχα μπορούμε να χρησιμοποιήσουμε άλλες ακολουθίες προκειμένου να τυπώσουμε ειδικούς χαρακτήρες ή να δώσουμε συγκεκριμένη μορφοποίηση στο κείμενό μας.

Παράδειγμα 1 Δοκιμάστε να τυπώσετε την ειδική ακολουθία \U0001F525. Τι τυπώνεται στην κονσόλα; Μπορείτε να βρείτε μια εκτενή λίστα με άλλες ειδικές ακολουθίες [εδώ](#) ενώ για να μάθετε περισσότερα μπορείτε να διαβάσετε για Unicode χαρακτήρες στην [wikipedia](#).

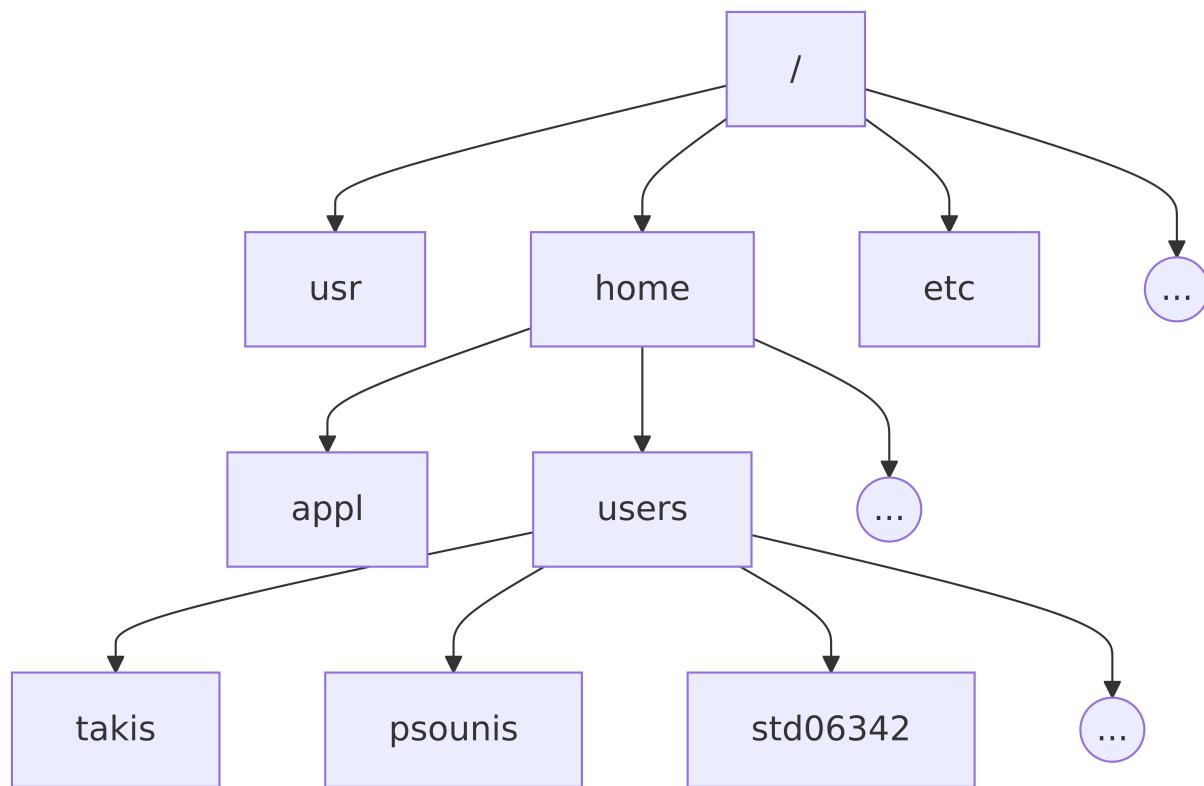
Παράδειγμα 2 Χρησιμοποιήστε την printf για να τυπώσετε την ακολουθία "Hello \033[1m world \033[0m !". Τι παρατηρείτε; Μπορείτε να φανταστείτε τι σημαίνουν οι ακολουθίες \033[1m και \033[0m; Τι συμβάνει αν αντικαταστήσετε την ακολουθία \033[1m με την ακολουθία \033[5m; Μπορείτε να μάθετε περισσότερα διαβάζοντας για [ANSI Escape codes](#).

Εργαστήριο #1: Unix και Git

Σκοπός του εργαστηρίου αυτού είναι να έλθουμε σε επαφή με: (1) τις βασικές εντολές του Unix και (2) το git, ένα σύστημα version control για την διαχείριση αρχείων κώδικα C.

Unix Tutorial

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:



Θες να ζωγραφίσεις και εσύ live κάποια από αυτά τα διαγράμματα; Μπορείς με το [mermaid](#).

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/stdXXXXXX`. 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 κατάλογοι:

- .. : 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 και θα τις χρησιμοποιούμε αναλόγως με το ζητούμενο στόχο.

Ένα ιδιαίτερα χρήσιμο κείμενο, με περαιτέρω τρόπους σύνταξης εντολών, πέρα από αυτές που θα αξιοποιήσουμε σήμερα, μπορούν να βρεθούν εδώ: <https://progintro.gitbook.io/assets/pdf/Unix.pdf>

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.

Σύνταξη:

`mkdir ονομα_καταλόγου`

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 `~/phw/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_ονομα.

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

Ο μεταγλωττιστής (compiler) 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 ονομα_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 πν χρησιμοποιείται για τη μετονομασία ή μετακίνηση αρχείων ή καταλόγων

Σύνταξη:

πν αρχικό_ονομα τελικό_όνομα

πν 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 ονομα_αρχείου`

Επιλογές:

- `-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 ονομα_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 1ο bit καθορίζει το δικαίωμα ανάγνωσης, το 2ο bit το δικαίωμα εγγραφής και το 3ο 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 (pipe) 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;)

Git και GitHub

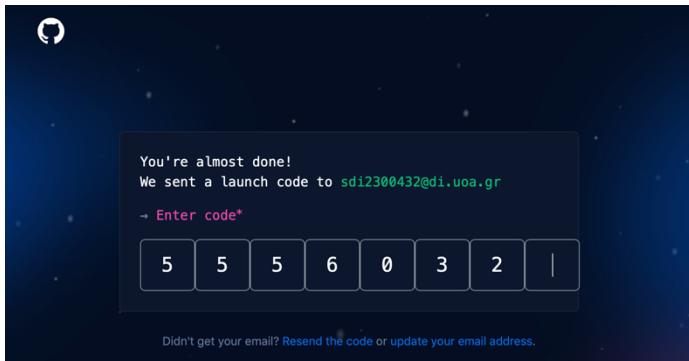
Για τις ανάγκες του μαθήματος, ένα από τα εργαλεία που θα χρησιμοποιήσουμε είναι το git καθώς και η cloud based υπηρεσία GitHub. Το git αποτελεί ένα εργαλείο που μας βοηθάει στη διαχείριση των εκδόσεων του πηγαίου κώδικα των προγραμμάτων που γράφουμε. Η υπηρεσία GitHub αποτελεί ένα on-line αποθετήριο για το git που είναι ελεύθερα διαθέσιμο.

Δημιουργία λογαριασμού στο GitHub

Για να αποκτήσουμε λογαριασμό στην υπηρεσία GitHub (αν δεν έχουμε ήδη) θα πρέπει να πραγματοποιήσουμε εγγραφή. Ανοίγουμε έναν browser και πληκτρολογούμε τη διεύθυνση: <https://github.com/join>

Στη φόρμα που θα εμφανιστεί συμπληρώνουμε τα στοιχεία μας, λύνοντας το πρόβλημα επιβεβαίωσης που θα μας ζητηθεί και πατάμε Create account.

Στη συνέχεια θα μας αποσταλεί ένα επιβεβαιωτικό email στη διεύθυνση που δηλώσαμε με έναν κωδικό επιβεβαίωσης τον οποίο θα πρέπει να εισάγουμε για να ολοκληρωθεί η δημιουργία του λογαριασμού μας.



Όταν εισάγουμε τον κωδικό επιβεβαίωσης ολοκληρώνεται η διαδικασία και ο λογαριασμός μας στο GitHub είναι έτοιμος προς χρήση.

Σύνδεση του λογαριασμού μας στο εργαστήριο με το λογαριασμό μας στο GitHub

Για να μπορούμε να χρησιμοποιήσουμε το GitHub ως αποθετήριο όταν εκτελούμε την εντολή git στα μηχανήματα του εργαστηρίου linux, θα πρέπει να δημιουργήσουμε ένα ζεύγος κλειδιών αυθεντικοποίησης και να το ορίσουμε ως έμπιστο στην υπηρεσία GitHub.

1. Δημιουργία προσωπικού κλειδιού

Η δημιουργία του κλειδιού γίνεται με την εντολή ssh-keygen, όπως φαίνεται παρακάτω

```

linux12:~>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key
(/home/users/sdi2300432/.ssh/id_rsa):
Created directory '/home/users/sdi2300432/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in .ssh/temp
Your public key has been saved in .ssh/temp.pub
The key fingerprint is:
SHA256:Pb9/DeknwU8cETcXME41Jyx0841g35W8kWB1TNjQFY thanassis@linux14
The key's randomart image is:
+---[RSA 3072]---+
|       o*XEX|
|       .++@*|
|       o+=0|
|       . . .=B|
| S o   o ++|
|       o. B |
|       ..B ..|
|       o.* +|
|       .o.= |
+---[SHA256]---+

```

2. Εκτύπωση προσωπικού κλειδιού

Εκτυπώνουμε τα περιεχόμενα του αρχείου **που περιέχει το public key** με την εντολή cat:

```
linux12:~>cat .ssh/id_rsa.pub
```

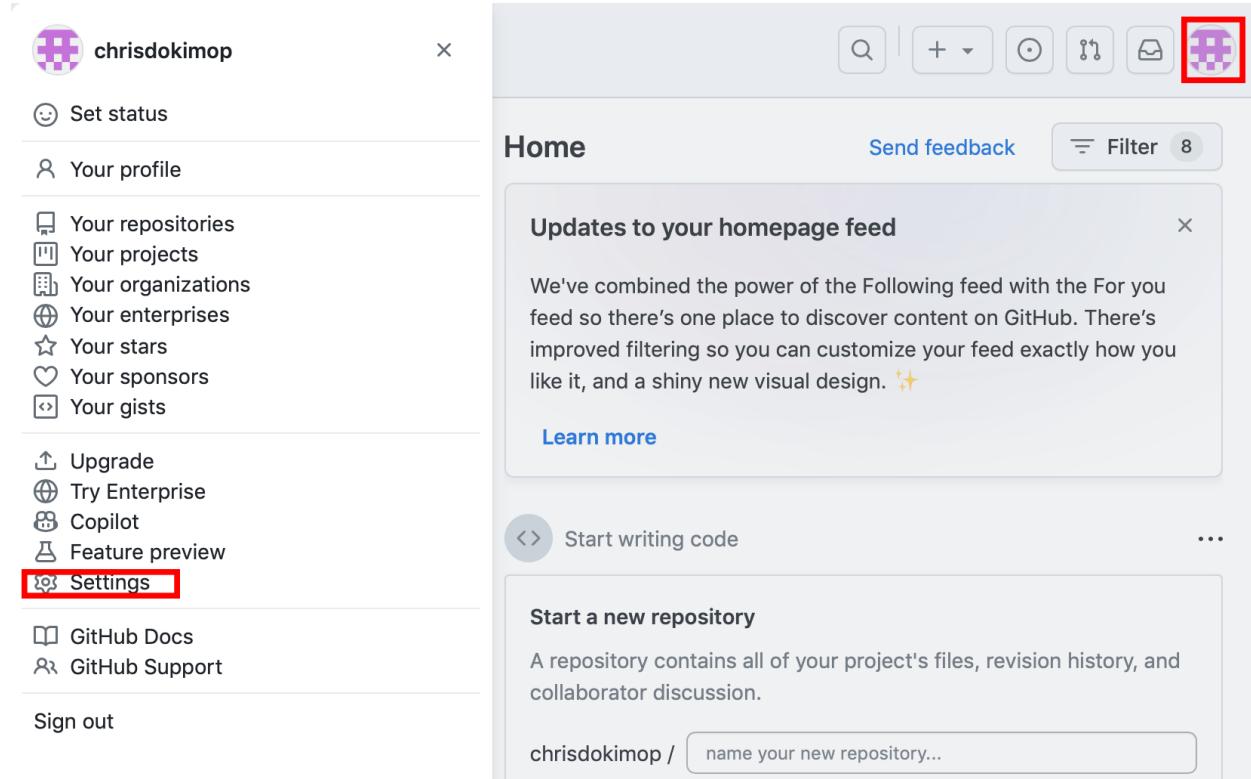
```

ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQgQDL
VHLGnzFEqkFqUizJQ/ILdmVtKpV8goIMo+v1b2AEySALQnfsvES1KaI0seIwatx1
CSwUzbUrE41yXK47PfU1vZ0D4H4MvsDuMiGLbuKAwogyKpc+Gx4EyNTsQqKvm21
/uYlH/p1B+b/6Pi7P7hwDUXN6PPmyViXAjrjFyb4mwbP1dxwDLYA0FwvMgb/Mk5q
6iJK02RhkQ0m1PtVdWcg+Y6i/VVLyj9gBZV0dCe1DPZZBZcIGkCC9ZLo35+LcDjt
frQC+hFN1Czdr0IUkHyAgtkXp8cR8ZAAUVr1sFXV1P9QX0TQqB6wWieclvj65t4s
3UW0BoYlrPRz+c4FIUVw4BID5Qt80WEUwlYmq4h+0actociwLg1CNUU2o820fmCH
IP1sU+YnSoFxUUUGwhimMnItWd7XD070c2M5w3o2ITaERdYwTZ9zFo1guj66DYnJ
XqIhr9KifpFY3XoI2D35gyMdWU2pr/ABrxGLV2e6avIvYZ8RNteQuQETSr+OTKM=
sdi2300432@linux12

```

3. Προσθήκη προσωπικού κλειδιού στο GitHub

Τώρα πρέπει να εισάγουμε το παραπάνω κλειδί στο GitHub. Όντας συνδεδεμένοι με το λογαριασμό που έχουμε στο github.com, πατάμε το εικονίδιο στην πάνω δεξιά γωνία, και στο μενού που θα ανοίξει επιλέγουμε το «Settings».



Στη συνέχεια επιλέγουμε το «SSH and GPG keys» και πατάμε την επιλογή «New SSH Key», όπως φαίνεται παρακάτω:

The screenshot shows the GitHub Settings interface for the user 'chrisdokimop'. The left sidebar has a red box around the 'SSH and GPG keys' link under the 'Access' section. The main content area has a red box around the 'New SSH key' button under the 'SSH keys' section.

chrisdokimop (chrisdokimop)
Your personal account

[Go to your personal profile](#)

[Public profile](#)
[Account](#)
[Appearance](#)
[Accessibility](#)
[Notifications](#)

Access

[Billing and plans](#)
[Emails](#)
[Password and authentication](#)
[Sessions](#)
SSH and GPG keys (selected)
[Organizations](#)
[Enterprises](#)
[Moderation](#)

Code, planning, and automation

[Repositories](#)

SSH keys

[New SSH key](#)

There are no SSH keys associated with your account.
Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

GPG keys

[New GPG key](#)

There are no GPG keys associated with your account.
Learn how to [generate a GPG key and add it to your account](#).

Vigilant mode

Τέλος συμπληρώνουμε ένα όνομα, κάνουμε copy-paste το κλειδί από το βήμα 2 και πατάμε «Add SSH Key»

- [Public profile](#)
- [Account](#)
- [Appearance](#)
- [Accessibility](#)
- [Notifications](#)

- Access**
- [Billing and plans](#)
- [Emails](#)
- [Password and authentication](#)
- [Sessions](#)
- SSH and GPG keys**
- [Organizations](#)
- [Enterprises](#)
- [Moderation](#)

- Code, planning, and automation**
- [Repositories](#)
- [Codespaces](#)
- [Packages](#)
- [Copilot](#)
- [Pages](#)
- [Saved replies](#)

- Security**
- [Code security and analysis](#)

Add new SSH Key

Title

Key type

Authentication Key

Key

```
ssh-rsa
AAAAAB3NzaC1yc2EAAAQABAAABgQDLVHLGnzFEqkFqUizJQ/lD
mVtKpV8goIMo+vlb2AEySALQnfsvES1KalOselwtx1CSwUzbUrE41yX
K47PfU1vZ0D4H4MvsDuMiGLbuKAwogyKpc+Gx4EyNTsQqKvM2I/u
YIH/p1B+b/6Pi7P7hwDUXN6PPmyViXAjrjFyb4mwBPldxwDLYAOFwvM
gb/Mk5q6iJKO2RhkQOm1PtVdWcg+Y6i/VVLy,9gBZVOdCeIDPZZBZcl
GkCC9ZLo35+LcDjfrQC+hFNICzdrOIUkHyAgtkXp8cR8ZAAUVr1sFX
VIP9QXOTQqb6wWieclvj65t4s3UW0BoYlrPRz+c4FIUVw4BID5Qt8O
WEUwlYmq4h+Oactociwlg1CNUU2o82OfmCHIPIsU+YnSoFxUUUGw
himMnlWd7XD07Oc2M5w3o2TaERdYwTZ9zFo1guj66DYnJXqlhr9Kif
pFY3Xo12D35gyMdWU2pr/ABrxGLV2e6avlyYZ8RNteQuQETSr+OTKM
= sdi2300432@linux12
```

Add SSH key

4. Ρύθμιση των παραμέτρων του git

Εκτελούμε τις παρακάτω εντολές, αντικαθιστώντας στα αντίστοιχα σημεία τη διεύθυνση email μας καθώς και το όνομά μας με λατινικούς χαρακτήρες:

```
git config --global user.email sdi2300432@di.uoa.gr
```

```
git config --global user.name "Christos Dokimopoulos"
```

Είμαστε πλέον έτοιμοι να χρησιμοποιήσουμε το git!

Άσκηση (ότι θα κάνουμε και στις ασκήσεις)

Πατήστε τον **ακόλουθο σύνδεσμο** για να δημιουργήσετε το δικό σας repository `progintro/lab01-YourGitHub` και πραγματοποιήστε τα ακόλουθα βήματα:

- Κατεβάστε το repository τοπικά τρέχοντας την εντολή:

```
$ git clone git@github.com:progintro/lab01-YourGitHub
Cloning into 'lab01-ethan42'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
```

```
remote: Compressing objects: 100% (2/2), done.  
Receiving objects: 100% (3/3), 5.10 KiB | 871.00 KiB/s, done.  
remote: Total 3 (delta 0), reused 2 (delta 0), pack-reused 0 (from 0)
```

2. Μέσα στον φάκελο lab01-YourGitHub που δημιούργησε η παραπάνω εντολή, προσθέστε ένα αρχείο info.txt με τα ακόλουθα στοιχεία:

Όνομα, Επώνυμο, sdiXXNNNNN(εφόσον έχετε)

3. Προσθέστε το αρχείο σας στο repository χρησιμοποιώντας την εντολή git add:

```
git add info.txt
```

4. Τρέξτε την εντολή git commit για να μονιμοποιήσετε τις αλλαγές σας:

```
git commit
```

5. Τρέξτε την εντολή git push για να ανεβάσετε τις αλλαγές σας στο GitHub.

```
$ git push  
Enumerating objects: 4, done.  
Counting objects: 100% (4/4), done.  
Delta compression using up to 4 threads  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 319 bytes | 63.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0)  
To github.com:progintro/lab01-ethan42  
 4ef457c..aa9f709 main -> main
```

Επιβεβαιώστε ότι βλέπετε τις αλλαγές πηγαίνοντας στο αντίστοιχο link του repository σας: <https://github.com/progintro/lab01-YourGitHub>.

Advanced: Περισσότερα παιχνίδια με την κονσόλα

Υπάρχουν πολλά online Unix/Linux/Ubuntu tutorials. Ένα από αυτά που είναι κάπως πιο δύσκολο είναι το [overthewire](#), το οποίο έχει μια σειρά από προκλήσεις σε capture-the-flag format (μιας μορφής παιχνιδιού) με σταδιακά αυξανόμενη δυσκολία. Σημείωση: μέχρι το επίπεδο 11 τα προβλήματα σε αυτό το site είναι εντός θέματος για το μάθημα, από εκεί και πέρα γίνονται σαφώς πιο δύσκολα / απαιτούν γνώσεις που θα αποκτήσουμε σε μεγαλύτερα έτη. Φυσικά αν θέλετε μπορείτε να τα εξερευνήσετε - πολλά από αυτά έχουν και λεπτομερέστατα writeups online.

Εργαστήριο #2: Editors

Προγραμματιστικά Περιβάλλοντα (IDEs)

Στο εργαστήριο αυτό, θα ασχοληθούμε με δύο προγραμματιστικά περιβάλλοντα για τη γλώσσα C: τον gcc μεταγλωτιστή της C σε περιβάλλον Linux, και ένα ολοκληρωμένο περιβάλλον ανάπτυξης IDE, όπως το Visual Studio Code (vscode). Επίσης, θα χρησιμοποιήσουμε και το πρόγραμμα scp/WinSCP για να μεταφέρουμε αρχεία από υπολογιστές Windows στον λογαριασμό μας στην σχολή (και αντίστροφα). Τέλος, θα χρησιμοποιήσουμε editors για να γράψουμε ένα πρόγραμμα που χειρίζεται δεδομένα εισόδου σε γλώσσα C, θα το μεταγλωτίσουμε, θα το εκτελέσουμε και θα πειραματιστούμε με την είσοδο και έξοδό του.

1. Το περιβάλλον προγραμματισμού Visual Studio Code

Στους λογαριασμούς των εργαστηρίων της σχολής είναι διαθέσιμο και το Visual Studio Code, που επίσης αποτελεί ένα ολοκληρωμένο γραφικό περιβάλλον ανάπτυξης κώδικα με διάφορες ευκολίες. Μπορούμε να το εγκαταστήσουμε στον προσωπικό μας υπολογιστή, ακολουθώντας τις οδηγίες που βρίσκονται εδώ:

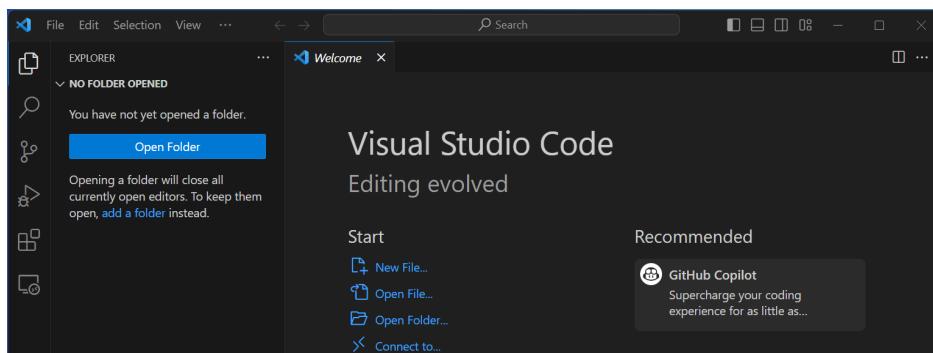
<https://k08.chatzi.org/vscode/>

Οι οδηγίες εγκατάστασης του vs code είναι υλικό που αναπτύχθηκε για το μάθημα Δομές δεδομένων (επόμενο εξάμηνο!) από τον καθ. Κωνσταντίνο Χατζηκοκολάκη τον οποίο και ευχαριστούμε ιδιαιτέρως!

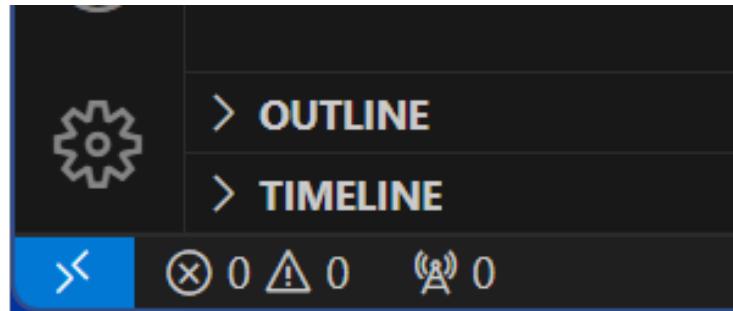
Προτού χρησιμοποιήσουμε το vscode για πρώτη φορά με το λογαριασμό μας στο εργαστήριο του τμήματος, θα πρέπει να έχουμε εκτελέσει την παρακάτω εντολή στη γραμμή εντολής του λογαριασμού μας μέσω κάποιου ssh client (π.χ., native ssh ή PuTTY):

```
curl https://k08.chatzi.org/vscode/config.sh | bash
```

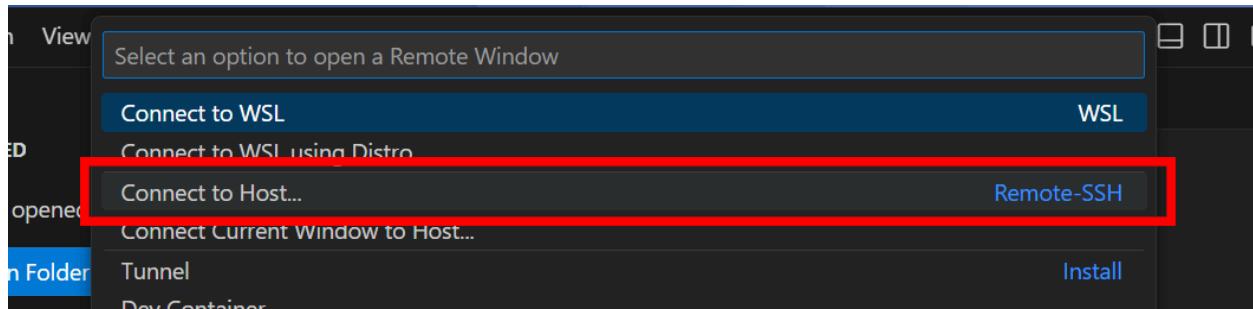
Γράφουμε code και εκτελούμε στην έναρξη ή επιλέγουμε το σχετικό εικονίδιο από το μενού προγραμμάτων για να ανοίξουμε το περιβάλλον προγραμματισμού Visual Studio Code.



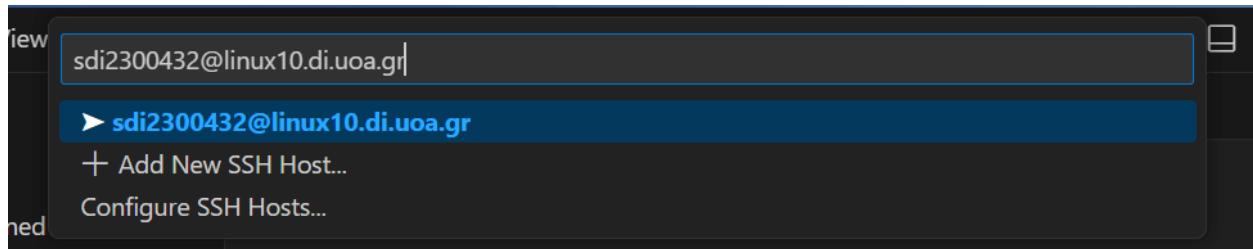
Για να συνδεθούμε με το λογαριασμό μας στο εργαστήριο Linux του Τμήματος, πατάμε το γαλάζιο εικονίδιο στην κάτω αριστερή γωνία του παραθύρου:



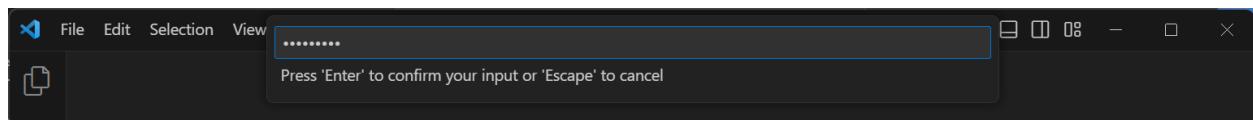
Στις επιλογές που θα ανοίξουν στην κορυφή του παραθύρου επιλέγουμε “Connect to Host...” (Remote SSH):



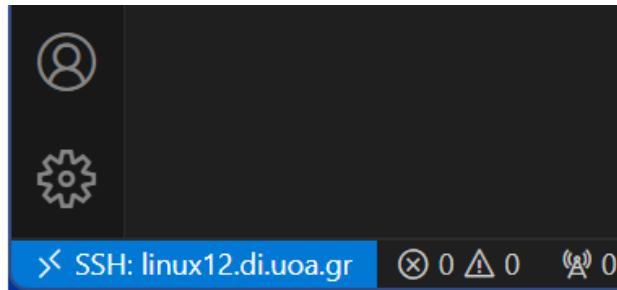
Στη συνέχεια πληκτρολογούμε το username μας ακολουθούμενο από @ και ένα από τα μηχανήματα του εργαστηρίου και πατάμε enter:



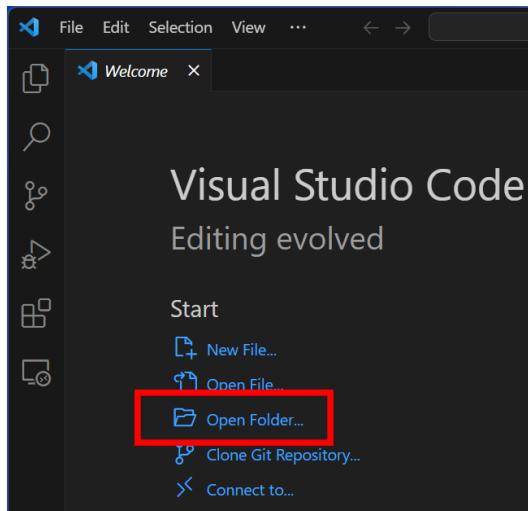
Θα μας ζητήσει τον κωδικό του λογαριασμού μας στο εργαστήριο Linux και ενδεχομένως, την πρώτη φορά που θα συνδεθούμε, αν εμπιστευόμαστε το κλειδί του συγκεκριμένου μηχανήματος.



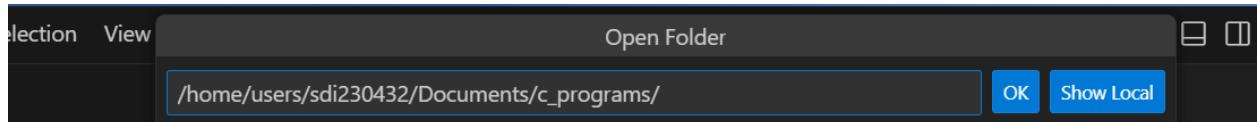
Αφότου βάλουμε τον κωδικό μας και πατήσουμε enter, θα πραγματοποιηθεί η σύνδεση. Όσο είμαστε συνδεδεμένοι με το μηχάνημα του εργαστηρίου Linux, αυτό θα εμφανίζεται στην κάτω αριστερή γωνία του παραθύρου στο γαλάζιο πλαίσιο:



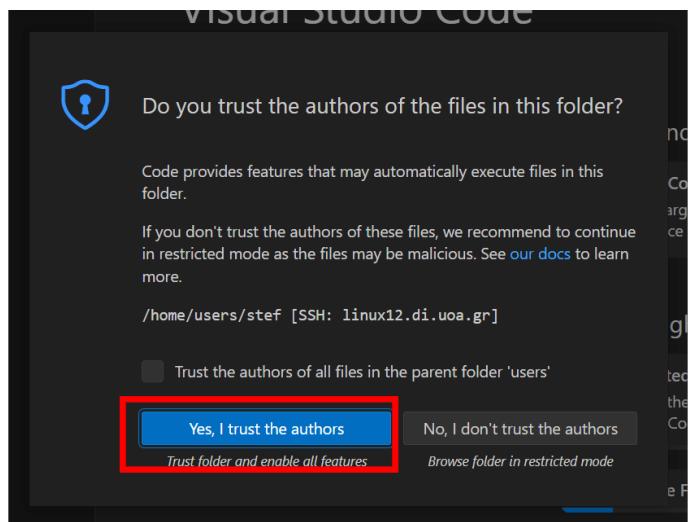
Από τις επιλογές του παραθύρου επιλέγουμε το Open Folder.... Σε περίπτωση που δεν είναι ανοιχτό το Welcome screen, η επιλογή αυτή υπάρχει και στο μενού File.



Πληκτρολογούμε η επιλέγουμε τον φάκελο Documents/c_programs που βρίσκεται μέσα στον προσωπικό μας κατάλογο και πατάμε enter.



Ενδεχομένως να μας ζητηθεί αν εμπιστευόμαστε το περιεχόμενο που βρίσκεται εκεί, πατάμε το Yes, I trust the authors.



Στο περιβάλλον που θα ανοίξει μπορούμε να περιηγηθούμε και να δοκιμάσουμε τα προγράμματά μας. Υπάρχουν κάποιες βασικές οδηγίες στο README.md που καλό είναι να τις διαβάσουμε, καθώς και μια πληθώρα ενδεικτικών προγραμμάτων από τις σημειώσεις του μαθήματος.

```

1 # Η εγκατάσταση ολοκληρώθηκε
2
3
4
5 # Οδηγίες χρήσης
6
7 - Με Ctrl-Shift-E (View / Explorer) εμφανίζονται όλα τα αρχεία που
   υπάρχουν στο Documents/c_programs directory.
8
9 - Κάντε κλικ στο example.c για να το ανοίξετε.
10
11 - Με Ctrl-` (View / Terminal) εμφανίζεται το τερματικό, στο οποίο μπορείτε
    να κάνετε μόνοι σας compile.
12
13 - Με Command-Shift-B (Terminal / Run Build Task) κάνετε αυτόματα compile
    το αρχείο που είναι ανοικτό εκείνη τη στιγμή.
14     - Αν υπάρχουν errors εμφανίζονται με κόκκινη γραμμή μέσα στον κώδικα.
15     - Αν δεν υπάρχουν errors το πρόγραμμα εκτελείται αφού μεταγλωτιστεί.
16
17 - Με F5 εκτελείτε το τρέχον αρχείο στον debugger. Η εκτέλεση διακόπτεται
    στην αρχή της main και μπορείτε να
    τη συνεχίσετε γραμμή-γραμμή, να βάλετε breakpoints, κλπ.
18
19
20 - Περισσότερες επιλογές για μεταγλώτιση/εκτέλεση υπάρχουν στο Terminal /
    Run task (πχ για προγράμματα που
    αποτελούνται από πολλά αρχεία .c).
21

```

Για να μεταγλωτίσουμε και να τρέξουμε το πρόγραμμα μας, Ctrl-Shift-B. 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.

The screenshot shows the Visual Studio Code interface connected via SSH to a Linux machine. The Explorer sidebar shows a project structure with files like .vscode, from_slides, multi_file_example, .gitignore, example, and README.md. The main editor window displays a C program named example.c. The code contains a printf statement and a closing brace. A red squiggle underlines the brace at line 9, column 44, indicating a syntax error. The terminal tab shows the output of a command to run the program, which failed with an exit code of 1 due to the syntax error.

Αντίστοιχα, αν το πρόγραμμά μας είναι σωστό, στην περιοχή του τερματικού θα εμφανιστούν τα αποτελέσματα της εκτέλεσής του.

2. Η εφαρμογή scp / WinSCP για μεταφορά αρχείων

Ένας εύκολος τρόπος να μεταφέρουμε τα αρχεία μας από τον υπολογιστή μας σε έναν απομακρυσμένο υπολογιστή (και τούμπαλιν) είναι η εντολή κονσόλας scp. Η εντολή scp συμπεριφέρεται όπως η εντολή cp σε συστήματα Linux, απλά αντί να αντιγράφει αρχεία τοπικά (μέσα στον ίδιο υπολογιστή) μπορεί να αντιγράφει από και προς απομακρυσμένους υπολογιστές. Προκειμένου να αντιγράψετε κάποιο αρχείο, ακουλουθούμε την σύνταξη:

```
scp local_file sdi2400999@linux08.di.uoa.gr:~/remote_file
```

όπου η εντολή παραπάνω αντιγράφει το τοπικό αρχείο με το όνομα local_file στο απομακρυσμένο αρχείο remote_file. Αντιθέτως, για να αντιγράψουμε ένα απομακρυσμένο αρχείο remote.txt σε ένα τοπικό local.txt ακολουθούμε την σύνταξη:

```
scp sdi2400999@linux08.di.uoa.gr:~/remote.txt local.txt
```

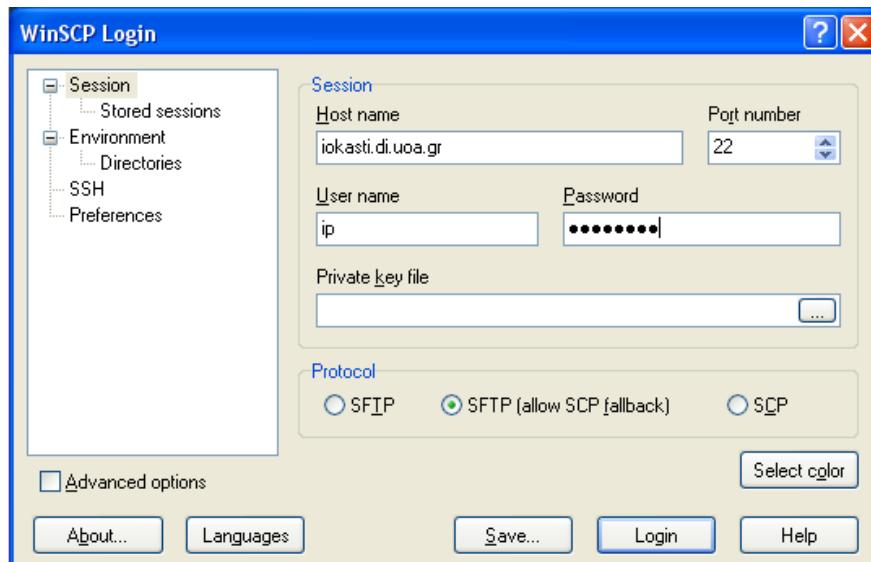
Η εφαρμογή scp είναι αποκλειστικά για χρήση μέσω κονσόλας. Στην συνέχεια θα χρησιμοποιήσουμε την εφαρμογή WinSCP που χρησιμοποιεί ένα Graphical User Interface (GUI) για την μεταφορά αρχείων από υπολογιστές Windows στους υπολογιστές των εργαστηρίων UNIX της σχολής. Με τον τρόπο αυτό, μπορούμε να χρησιμοποιούμε 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>

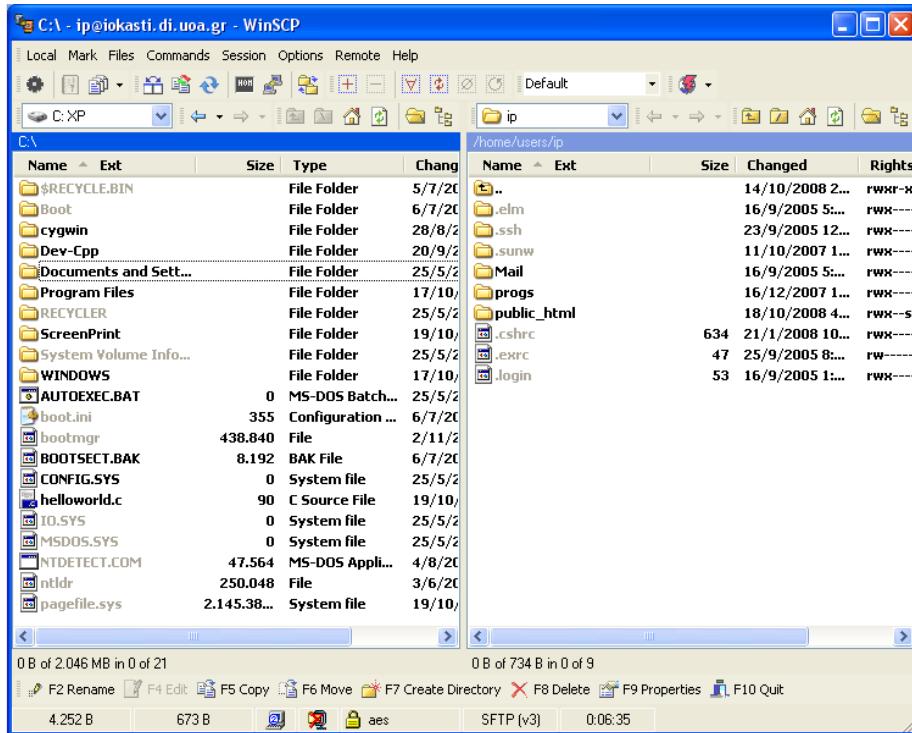
Εκτελώντας το πρόγραμμα βλέπουμε την ακόλουθη οθόνη:



όπου πληκτρολογούμε τα στοιχεία σύνδεσης μας δηλαδή:

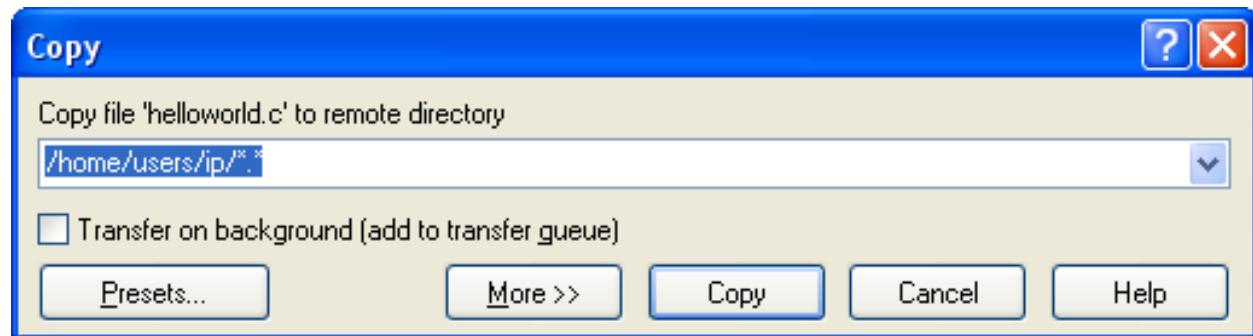
- Τον υπολογιστή που θα συνδεθούμε (http://cgi.di.uoa.gr/~ip/linux_lab_machines.html)
- Το όνομα χρήστη
- Τον κωδικό μας

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:

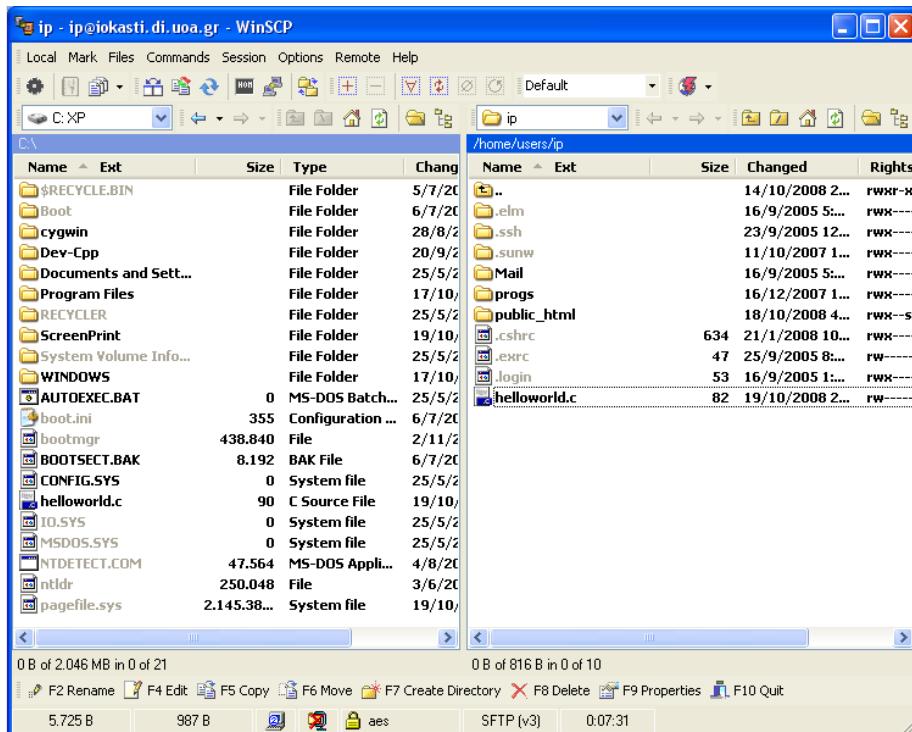


Με αυτό το μήνυμα ζητείται η επιβεβαίωση μας για την μεταφορά του αρχείου από τον τοπικό κατάλογο στον χώρο του λογαριασμού μας της σχολής. Αν πατήσουμε Copy το αρχείο μεταφέρεται στον λογαριασμό μας.

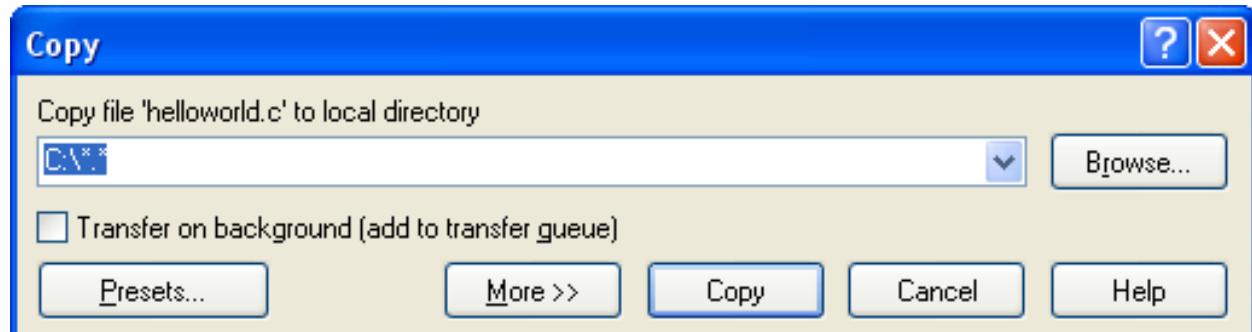
Βεβαίως είναι εφικτό να ακολουθήσουμε και την αντίστροφη διαδικασία, για να αντιγράψουμε αρχεία από τον λογαριασμό μας στην σχολή, στον τοπικό δίσκο.

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:



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



όπως μετονομασία των αρχειων, 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 Unix και VS Code

Κάνουμε login σε έναν από τους υπολογιστές του εργαστηρίου μας μέσω VS Code όπου θα δημιουργήσουμε ένα καινούριο αρχείο με όνομα `readint.c` και θα εισάγουμε τα ακόλουθα περιεχόμενα:

```
/* File: readint.c */

#include <stdio.h>

int main() {
    int number;

    printf("Please give me a number: ");
    fflush(stdout);
    scanf("%d", &number);

    printf("I just read this number: %d\n", number);
    return 0;
}
```

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

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

όπου γίνεται 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.

Μεταγλωτίστε και τρέξτε το παραπάνω πρόγραμμα. Πως συμπεριφέρεται;

Τι συμβαίνει αν αντί για αριθμό δώσετε μια λέξη ή μια κενή γραμμή;

Τι συμβαίνει αν αντί για αριθμό πληκτρολογήσουμε `Ctrl+D` (EOF);

Τι θα συμβεί αν αφαιρέσουμε την εντολή `fflush(stdout)`;

Γιατί να μην δώσουμε μια σταθερά τιμή στην μεταβλητή `number` στο πρόγραμμά μας - γιατί να ζητήσουμε από τον χρήστη να δώσει μια τιμή;

Άσκηση (ίσως μας βοηθήσει και στην Εργασία #0)

Γράψτε ένα πρόγραμμα C `calc.c` που δρα ως ένα κομπιουτεράκι (calculator) για να πραγματοποιεί πρόσθεση. Το πρόγραμμά σας πρέπει να ζητά δύο ακεραίους από την πρότυπη είσοδο (`stdin`) και στην συνέχεια να τυπώνει το άθροισμά τους. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./calc
Please enter the first number: 42
Please enter the second number: 58
The sum of the two numbers is: 100
$ echo $?
0
```

Advanced #1: Calculator Υψηλής Ακρίβειας

Τροποποιήστε το παραπάνω πρόγραμμα προκειμένου να μπορεί να δέχεται μεγάλους αριθμούς μέχρι και 1018. Το πρόγραμμά σας συνεχίζει να παράγει τα αναμενόμενα αποτελέσματα;

Advanced #2: Calculator με Διάφορες Πράξεις

Τροποποιήστε το calc πρόγραμμά σας προκειμένου να πραγματοποιεί πρόσθεση Ή αφαίρεση ανάλογα με το τι προτιμάει ο χρήστης. Πως θα σχεδιάζατε την εφαρμογή σας;

Advanced #3: Calculator που χειρίζεται σφάλματα

Τροποποιήστε το calc πρόγραμμα προκειμένου να βγάζει κωδικό εξόδου 1 εάν οτιδήποτε πάει στραβά κατά την εκτέλεση του προγράμματος - για παράδειγμα εάν ο χρήστης δώσει το string "hello" αντί για αριθμό.

ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 1η)

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”.

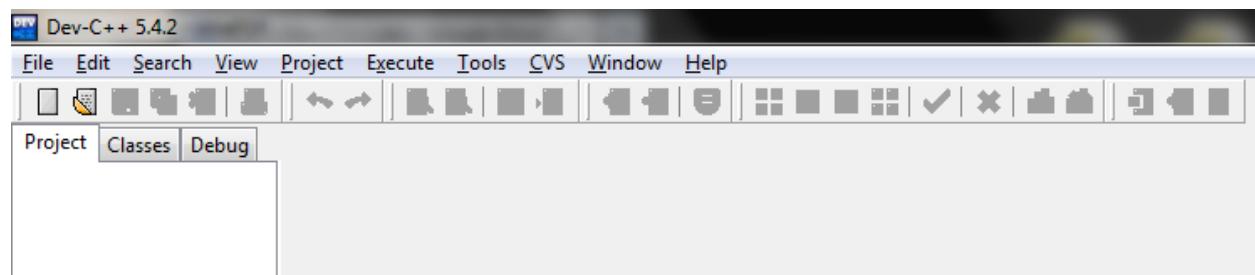
Οπότε, είναι εμφανές ότι το να βασιζόμαστε μόνο στα μηνύματα του μεταγλωττιστή για να διορθώσουμε τα συντακτικά λάθη δεν είναι μια τακτική που αποδίδει πάντα. Αυτό που χρειάζεται είναι εμπειρία ώστε να αποκτηθεί εξοικείωση με τα διάφορα συντακτικά λάθη, αλλά και προσοχή κατά τη συγγραφή των κώδικα ώστε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

Παράρτημα Β'. Το περιβάλλον προγραμματισμού Dev-C++

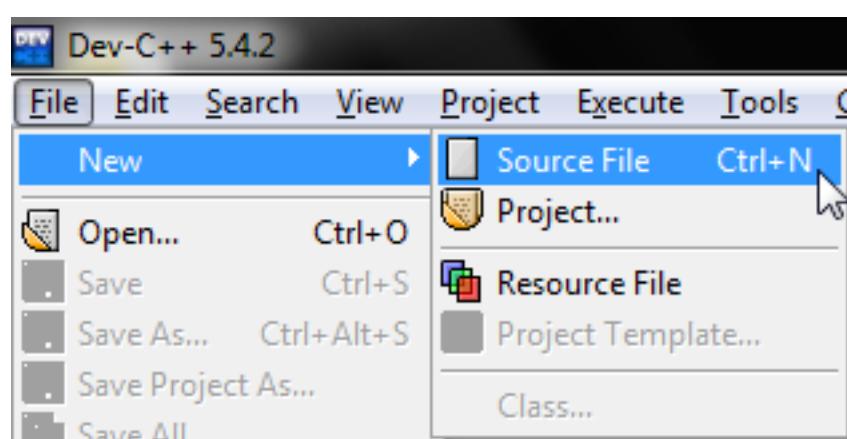
Στους λογαριασμούς των Windows συστημάτων της σχολής είναι διαθέσιμο το IDE Dev-C++, ένα γραφικό περιβάλλον ανάπτυξης κώδικα που απλοποιεί τη διαδικασία συγγραφής και μεταγλωττισης κώδικα (“αποκρύπτοντάς” μας την ύπαρξη και λειτουργία του gcc μεταγλωττιστή). Μπορούμε να το εγκαταστήσουμε στον προσωπικό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

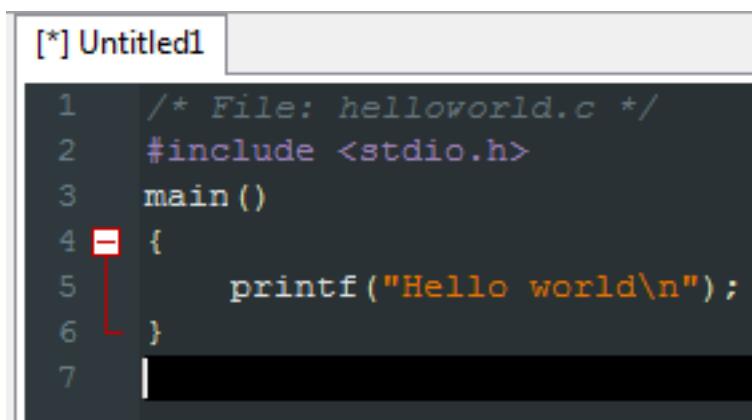
Ας γράψουμε εδώ λοιπόν, το πρώτο μας πρόγραμμα C:

```
/* 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 αστέρι στα αριστερά της καρτέλας υποδηλώνει πως έχουν γίνει αλλαγές που δεν έχουν σωθεί).



The screenshot shows a terminal window titled "Untitled1". The code inside is:

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

A red box highlights the opening brace of the main function, and a red arrow points from it to the closing brace, indicating a syntax error.

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 Εργοσίας) όπου μας ενδιαφέρει να αποθηκεύσουμε το αρχείο. Τέλος, δίνουμε ένα κατάλληλο όνομα στο αρχείο (π.χ. helloworld.c) και αποθηκεύουμε.

Για να μεταγλωτίσουμε 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 μεταγλωτιστή, τα οποία περιγράφουν τη φύση του προβλήματος και μας καθοδηγούν για την επίλυσή του. Αφού εκτελέσουμε το μεταγλωτιστή, τότε θα εμφανιστούν μηνύματα όπως στην ακόλουθη εικόνα.

The screenshot shows the Dev-C++ IDE interface. The menu bar includes File, Edit, Search, View, Project, Execute, Tools, CVS, Window, and Help. The toolbar contains various icons for file operations like Open, Save, and Build. The Project tab is selected in the left sidebar, which is currently empty. The main code editor window displays the following C code:

```

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

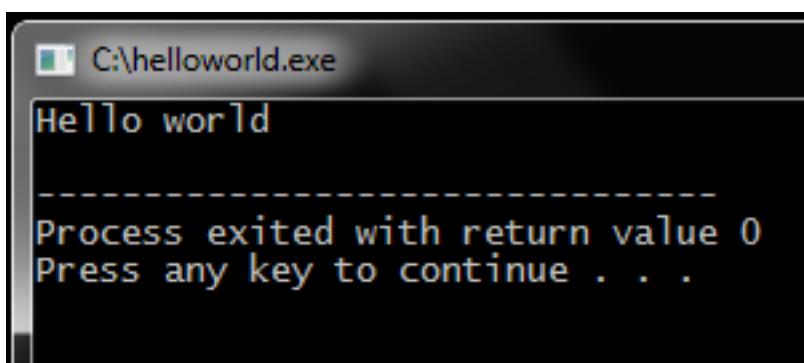
```

The closing brace at line 6 is highlighted in red, indicating a syntax error. Below the code editor, the file name "helloworld.c" is shown. At the bottom, there is a status bar with tabs for Compiler (2), Resources, Compile Log, Debug, Find Results, and a warning icon.

Line	Col	Message
		In function 'main':
6	1	[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.



To 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.

Εργαστήριο #3: Δομές Επανάληψης

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 γράψουμε συναρτήσεις προκειμένου να λύσουμε προγραμματιστικά προβλήματα.

Δομές επανάληψης while και do...while:

WHILE	DO...WHILE
while (E ₂) { }	do { } while (E ₂);
E ₂ : Συνθήκη Εισόδου και Συνέχισης Επανάληψης	E ₂ : Συνθήκη Συνέχισης Επανάληψης

Η δομή επανάληψης for:

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

Ισοδύναμη δομή while με την δομή for:

FOR	WHILE
for (E ₁ ; E ₂ ; E ₃) { }	E ₁ ; while (E ₂) { E ₃ ; }

printf: Σύμβολα Ακεραίων Σύμβολα Κινητής Υποδιαστολής

d: int

ld: long

lld: long long

f: float ή double

u: unsigned int

Lf: long double

lu: unsigned long

llu: unsigned long long

Άσκηση 1: Υπολογισμός Αθροίσματος Σειράς (seq.c)

Σε αυτήν την άσκηση, θα γράψουμε ένα πρόγραμμα seq.c το οποίο θα υπολογίζει διάφορα αθροίσματα σειράς. Για κάθε ένα από αυτά, θα φτιάχνουμε μια διαφορετική συνάρτηση υπολογισμού.

1.1. Γράψτε μια συνάρτηση sum που να υπολογίζει το άθροισμα της σειράς:

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

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

1.2. Γράψτε μια συνάρτηση basel που να υπολογίζει το άθροισμα της σειράς:

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

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

Τι συμβαίνει αν αυξήσετε τους όρους; Ρίξτε μια ματιά στο [Basel Theorem](#) και ελέγχτε αν τα αποτελέσματά σας συμφωνούν με τα μαθηματικά.

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

1.3. Γράψτε μια συνάρτηση pi_approx που να προσεγγίζει το π χρησιμοποιώντας την έκφραση:

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

χρησιμοποιώντας τους 100 πρώτους όρους της σειράς και να τυπώνει το αποτέλεσμα. Να χρησιμοποιηθεί η δομή επανάληψης do...while. Για την υλοποίηση θα χρειαστείτε την συνάρτηση sqrt (square root).

```
double sqrt(double)
```

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

Επιστρέφει την τετραγωνική ρίζα του αριθμού που δέχεται σαν όρισμα.

Στην μεταγλώττιση με τον `gcc` απαιτείται και η επιλογή `-lm` για να συμπεριληφθεί η μαθηματική βιβλιοθήκη.

1.3.1. Τροποποίηση της `ri_approx` μέχρι όριο ακρίβειας 10^{-15} . Παρατηρήστε ότι οι όροι που προστίθενται στην σειρά γίνονται ολοένα και 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} .

1.3.2. Τροποποίηση της `ri_approx` ώστε να τυπώνει 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);
```

όπου:

- a: Ορίζουμε το πλήθος των θέσεων που θα γίνει η εκτύπωση.
- b: Ορίζουμε το πλήθος των ψηφίων μετά την υποδιαστολή που θα εκτυπωθούν.

Μέχρι πόσα ψηφία του π μπορείτε να υπολογίσετε / τυπώσετε; Αυτό το πρόβλημα έχει τυραννήσει γενιές και γενιές στα μαθηματικά και αργότερα - όταν οι αριθμοί έγιναν μεγάλοι - στην πληροφορική [1] [2].

1.4. Υλοποιήστε μια συνάρτηση `eta_two` η οποία να υπολογίζει το άθροισμα της σειράς:

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

ή αλλιώς γραμμένη ως:

$$S_1 = \sum_{i=1}^{\infty} \frac{(-1)^{n-1}}{n^2}$$

Η συνάρτησή σας θέλουμε να ζητάει από τον χρήστη το πλήθος των όρων που θα χρησιμοποιήσει και να τυπώνει το αποτέλεσμα με ακρίβεια 6 δεκαδικών ψηφίων. Μπορείτε να χρησιμοποιήσετε όποια δομή επανάληψης επιθυμείτε.

Φαίνεται το αποτέλεσμά σας να συγκλίνει σε κάποιον αριθμό; Μπορείτε να το εξηγήσετε με την βοήθεια της συνάρτησης `Dirichlet eta`;

1.5. Εξάσκηση σε Σειρές

Μπορείτε να κάνετε περισσότερη εξάσκηση, υλοποιώντας τις ακόλουθες συναρτήσεις στο πρόγραμμά `seq.c`:

- Συνάρτηση `harmonic` [3] που υπολογίζει το άθροισμα:

$$S_2 = \sum_{i=1}^{\infty} \frac{(-1)^{n+1}}{n} = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$$

- Συνάρτηση leibniz [4] που να υπολογίζει το άθροισμα:

$$S_3 = \sum_{i=1}^{\infty} \frac{(-1)^{n+1}}{2n+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

- Συνάρτηση zeta_four [5] που να υπολογίζει το άθροισμα:

$$S_6 = \sum_{i=1}^{\infty} \frac{1}{n^4} = \frac{1}{1^4} + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \dots$$

- Συνάρτηση wallis [6] που να υπολογίζει το γινόμενο:

$$P = \prod_{i=1}^{\infty} \frac{2n}{2n-1} \cdot 2n2n+1 = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \dots$$

Άσκηση 2: Υπολογισμός ριζών τριωνύμου (root.c)

2.1. Παραγωγή τυχαίων αριθμών

Γράψτε ένα πρόγραμμα root.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.

```
int rand()
```

Απαιτείται ενσωμάτωση του αρχείου επικεφαλίδας stdlib.h

Επιστρέφει έναν ψευδο-τυχαίο ακέραιο από 0 έως RAND_MAX - συμβολική σταθερά που ορίζεται στο αρχείο επικεφαλίδας stdlib.h

```
rand(unsigned int seed)
```

Απαιτείται ενσωμάτωση του αρχείου επικεφαλίδας stdlib.h

Αρχικοποιεί την γεννήτρια ψευδο-τυχαίων αριθμών με βάση τον αριθμό/φύτρα seed. Συνήθως χρησιμοποιούμε σα φύτρα την τρέχουσα ώρα, όπως επιστρέφεται από την κλήση της συνάρτησης time(NULL) (απαιτείται το αρχείο επικεφαλίδας time.h)

2.2. Εύρεση ριζών

Τροποποιήστε το `root.c` ώστε 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.

Η δομή ελέγχου `if ... else`

```
if (C)
    E1
else
    E2
```

Εάν η συνθήκη `C` είναι αληθής, τότε εκτελείται η εντολή (ή μπλοκ εντολών) `E1`, αλλιώς η εντολή (ή block εντολών) `E2`

2.3. Μιγαδικές Ρίζες

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 (`birthdate.c`)

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

Έστω η ημερομηνία $DD/MM/YYYY$ (δύο ψηφία για την μέρα DD - Day, δύο ψηφία για τον μήνα MM - Month, 4 ψηφία για τον χρόνο $YYYY$ - Year).

Αν $MM \leq 2$, τότε:

$$\begin{aligned} NYYYY &= YYYY - 1 \\ NMM &= 0 \end{aligned}$$

Αν $MM > 2$, τότε:

$$\begin{aligned} NYYYY &= YYYY \\ NMM &= \left\lfloor \frac{4 \cdot MM + 23}{10} \right\rfloor \end{aligned}$$

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

Για τον υπολογισμό της ημέρας, αν:

- $IDAY \bmod 7 = 0$, τότε DAY = Saturday
- $IDAY \bmod 7 = 1$, τότε DAY = Sunday
- ...
- $IDAY \bmod 7 = 6$, τότε DAY = Friday

Όπου, με $\lfloor x \rfloor$ συμβολίζουμε τον μέγιστο ακέραιο αριθμό που δεν είναι μεγαλύτερος του αριθμού x - γνωστή και ως συνάρτηση floor ([αντίστοιχα υπάρχει και η συνάρτηση ceil](#)). Δεν είμαστε σίγουροι πως λειτουργεί - πως θα το βρούμε;

Εργασία: Γράψτε ένα πρόγραμμα birthdate.c το οποίο να χρησιμοποιεί τον αλγόριθμο παραπάνω και να υπολογίζει τι ημέρα γεννηθήκατε (ενσωματώστε την ημερομηνία γέννησής σας μέσα στο πρόγραμμα).

Ο παραπάνω υπολογισμός φαίνεται περίπλοκος; Δεν έχετε δει τίποτε ακόμη! Πραγματικές συναρτήσεις βιβλιοθήκης ημερομηνιών πρέπει να χειρίζονται ημερομηνίες και στο παρελθόν, με ιδιαίτερα περίπλοκες αλλαγές - τι γίνεται για παράδειγμα αν ο χρήστης θέλει να βρει την ημερομηνία 16 Φεβρουαρίου 1923 στην Ελλάδα [7];

ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 2^η)

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 πώς 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 επιχειρεί να κάνει αυτή τη δουλειά είναι το εξής:

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

int 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 τέτοιους 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 πώς 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 εμφανίσουμε τις τιμές τους με σωστή τοποθέτηση συναρτήσεων `printf` ως εξής:

```
int 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:

```

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 και των όρων γίνεται σωστά. Άρα το πρόβλημα πρέπει να βρίσκεται στο πλήθος των όρων που χρησιμοποιούμε, το οποίο, 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 ότι ο τρέχων όρος πρέπει να είναι μικρότερος ή ίσος 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, αλλά σε κάποια παραδείγματα δεν θα λειτουργεί σωστά. Τότε, αυτό που έχει συμβεί είναι ότι δεν έχουμε καλύψει όλες τις πιθανές περιπτώσεις, κάτι που είναι υποχρεωτικό 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 κάποιο από τα παραδείγματα που δοκιμάσαμε δεν δίνει το αναμενόμενο αποτέλεσμα, τότε έχουμε παραλείψει να καλύψουμε αυτή την περίπτωση στον κώδικά μας. Πρέπει να σκεφτούμε ποια είναι τα χαρακτηριστικά του συγκεκριμένου παραδείγματος που το κάνουν να μην εμπίπτει σε όσα έχουμε σκεφτεί και υλοποιήσει. Είναι πολύ πιθανό να χρειαστεί να καταφύγουμε ξανά στις τεχνικές που περιγράψαμε πριν και να τοποθετήσουμε κατάλληλες `printf` σε κρίσιμα σημεία του προγράμματος. Μόλις βρούμε τι φταίει, εμπλουτίζουμε την αρχική μας σκέψη ώστε να καλύψει και αυτήν την περίπτωση και κάνουμε τις απαραίτητες προσθήκες στον κώδικά μας.

Γενικά, η διόρθωση λογικών λαθών απαιτεί εμπειρία, υπομονή και εξοικείωση με το πρόβλημα και τον κώδικα 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 κώδικα 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 στη διαδικασία αποσφαλμάτωσης. Είναι σημαντικό να εξοικειωθείτε με τη χρήση κάποιου τέτοιου προγράμματος, π.χ. με τον gdb που βρίσκεται εγκατεστημένος στα μηχανήματα Linux του Τμήματος. Παρουσίαση του gdb θα γίνει σε επόμενο εργαστήριο.

Άσκηση Αποσφαλμάτωσης (limit.c)

Το παρακάτω πρόγραμμα limit.c ευελπιστεί να υπολογίζει την παράσταση $1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \frac{1}{16} - \dots$ 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 $\frac{1}{2}$. 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 και να γράψετε ένα σωστό limit.c;

```
#include <stdio.h>

#define LIMIT 0.008

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

    while (c < LIMIT) {
        S = S + c;
        sign = (-1) * sign;
        c = sign * (1.0 / 2) * c;
    }

    printf("%f\n", S);
}
```

Εργαστήριο 4: Είσοδος και Έξοδος

Είσοδος/Έξοδος 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: Εκτύπωση Χαρακτήρων (printchar.c)

Γράψτε ένα πρόγραμμα printchar.c που να εκτυπώνει τους χαρακτήρες με ASCII κωδικό που είναι πολλαπλάσιο του 3, και μεταξύ 33 και 105, μαζί με τους κωδικούς τους σε δεκαδική και δεκαεξαδική μορφή. Το πρόγραμμά σας θέλουμε να τυπώνει έναν χαρακτήρα ανά γραμμή. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./printchar
! - (dec: 33, hex: 21)
$ - (dec: 36, hex: 24)
' - (dec: 39, hex: 27)
* - (dec: 42, hex: 2a)
- - (dec: 45, hex: 2d)
0 - (dec: 48, hex: 30)
3 - (dec: 51, hex: 33)
...
```

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	000	0x00	(sp)	32	040	0x20	@	64	100	0x40	'	96	140	0x60
(soh)	1	001	0x01	!	33	041	0x21	A	65	101	0x41	a	97	141	0x61
(stx)	2	002	0x02	"	34	042	0x22	B	66	102	0x42	b	98	142	0x62
(etx)	3	003	0x03	#	35	043	0x23	C	67	103	0x43	c	99	143	0x63
(eot)	4	004	0x04	\$	36	044	0x24	D	68	104	0x44	d	100	144	0x64
(enq)	5	005	0x05	%	37	045	0x25	E	69	105	0x45	e	101	145	0x65
(ack)	6	006	0x06	&	38	046	0x26	F	70	106	0x46	f	102	146	0x66
(bel)	7	007	0x07	'	39	047	0x27	G	71	107	0x47	g	103	147	0x67
(bs)	8	010	0x08	(40	050	0x28	H	72	110	0x48	h	104	150	0x68
(ht)	9	011	0x09)	41	051	0x29	I	73	111	0x49	i	105	151	0x69
(nl)	10	012	0x0a	*	42	052	0x2a	J	74	112	0x4a	j	106	152	0x6a
(vt)	11	013	0x0b	+	43	053	0x2b	K	75	113	0x4b	k	107	153	0x6b
(np)	12	014	0x0c	,	44	054	0x2c	L	76	114	0x4c	l	108	154	0x6c
(cr)	13	015	0x0d	-	45	055	0x2d	M	77	115	0x4d	m	109	155	0x6d
(so)	14	016	0x0e	.	46	056	0x2e	N	78	116	0x4e	n	110	156	0x6e

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(si)	15	017	0x0f	/	47	057	0x2f	O	79	117	0x4f	o	111	157	0x6f
(dle)	16	020	0x10	0	48	060	0x30	P	80	120	0x50	p	112	160	0x70
(dc1)	17	021	0x11	1	49	061	0x31	Q	81	121	0x51	q	113	161	0x71
(dc2)	18	022	0x12	2	50	062	0x32	R	82	122	0x52	r	114	162	0x72
(dc3)	19	023	0x13	3	51	063	0x33	S	83	123	0x53	s	115	163	0x73
(dc4)	20	024	0x14	4	52	064	0x34	T	84	124	0x54	t	116	164	0x74
(nak)	21	025	0x15	5	53	065	0x35	U	85	125	0x55	u	117	165	0x75
(syn)	22	026	0x16	6	54	066	0x36	V	86	126	0x56	v	118	166	0x76
(etb)	23	027	0x17	7	55	067	0x37	W	87	127	0x57	w	119	167	0x77
(can)	24	030	0x18	8	56	070	0x38	X	88	130	0x58	x	120	170	0x78
(em)	25	031	0x19	9	57	071	0x39	Y	89	131	0x59	y	121	171	0x79
(sub)	26	032	0x1a	:	58	072	0x3a	Z	90	132	0x5a	z	122	172	0x7a
(esc)	27	033	0x1b	;	59	073	0x3b	[91	133	0x5b	{	123	173	0x7b
(fs)	28	034	0x1c	<	60	074	0x3c	\	92	134	0x5c		124	174	0x7c
(gs)	29	035	0x1d	=	61	075	0x3d]	93	135	0x5d	}	125	175	0x7d
(rs)	30	036	0x1e	>	62	076	0x3e	^	94	136	0x5e	~	126	176	0x7e
(us)	31	037	0x1f	?	63	077	0x3f	_	95	137	0x5f	(del)	127	177	0x7f

Άσκηση 2: Κατασκευή Πυραμίδας (pyramid.c)

2.1. Γράψτε ένα πρόγραμμα pyramid.c που να εκτυπώνει στην οθόνη σχήμα με την ακόλουθη μορφή χρησιμοποιώντας αποκλειστικά την συνάρτηση putchar:

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



Το πλήθος των γραμμών που θα εκτυπωθούν να το διαβάζετε με χρήση scanf.

```
int putchar(int c)
```

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. Τροποποιήστε το pyramid.c ώστε να εκτυπώνει στην οθόνη σχήμα με την ακόλουθη μορφή:

```
*
```

```
***
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```



Extra: τροποποιήστε το `pyramid.c` ώστε ο αριθμός με τον οποίο ανξάνει τον αριθμό των * ανά γραμμή να καθορίζεται επίσης από τον χρήστη με χρήση `scanf`.



Άσκηση 3: Τροποποίηση Κειμένου (lowercase.c)

3.1 Αντιγράψτε το αρχείο `capitalize.c` παρακάτω, μεταγλωττίστε το και εκτελέστε το με είσοδο το ίδιο το πηγαίο αρχείο `capitalize.c`. Παρατηρήστε τη λειτουργία του.

```
/* File: capitalize.c */
```

```
#include <stdio.h>
```

```
int 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 */
```

```
    }
```

```
    return 0;
```

```
}
```



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 Γράψτε ένα 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 `lowercase.c`, ώστε να μετατρέπονται ταυτόχρονα και οι κεφαλαίοι χαρακτήρες σε μικρούς και οι μικροί σε κεφαλαίους.



Άσκηση 4: Κωδικοποίηση και Αποκωδικοποίηση Κειμένου (encode.c και decode.c)

Σε αυτήν την άσκηση θα γράψουμε προγράμματα για την δεκαεξαδική κωδικοποίηση και αποκωδικοποίηση ενός κειμένου.

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

4.2 Γράψτε ένα πρόγραμμα `encode.c` που να διαβάζει από την είσοδο χαρακτήρες και, για καθένα απ' αυτούς, να εκτυπώνει στην έξοδο τον ASCII κωδικό που έχει σε δεκαεξαδική μορφή χρησιμοποιώντας ακριβώς δύο ψηφία. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ echo hello world | ./encode  
68656c6c6f20776f726c640a
```

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

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

4.5 Γράψτε ένα πρόγραμμα `decode.c` που να διαβάζει από την είσοδο τα δεκαεξαδικά ψηφία για τον κάθε χαρακτήρα και για κάθε δύο από αυτά τα ψηφία να εκτυπώνει τον χαρακτήρα που αντιστοιχεί σε αυτόν τον δεκαεξαδικό αριθμό. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ echo -ne 68656c6c6f20776f726c640a | ./decode  
hello world
```

Ο αριθμός 68 αντιστοιχεί στον χαρακτήρα 'h' κοκ. Για την αποκωδικοποίηση μπορείτε να χρησιμοποιήσετε την συνάρτηση `getchar()` ή την `scanf("%02x", ...)`.

4.6 Μεταγλωττίστε το και εκτελέστε το με ανακατεύθυνση της εισόδου από το αρχείο `encoded_text`.

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 `encoded_text`. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ echo hello world | ./encode | ./decode  
hello world
```

Εργαστήριο #5: Συναρτήσεις και Αναδρομή

Συναρτήσεις και Αναδρομή

Στο εργαστήριο αυτό θα συνεχίσουμε την εξοικείωσή μας με την χρήση συναρτήσεων και θα υλοποιήσουμε τις πρώτες μας αναδρομικές συναρτήσεις, δηλαδή συναρτήσεις που στο σώμα τους καλούν τον εαυτό τους.

Άσκηση 1. Η εικασία του Collatz (collatz.c)

Η εικασία Collatz είναι ένα από τα πιο διάσημα άλυτα προβλήματα των μαθηματικών. Πολλοί μαθηματικοί έχουν επιχειρήσει κατά καιρούς να την αποδείξουν και μέχρι στιγμής δεν έχουμε ακόμα μια ευρέως αποδεκτή απόδειξη. Η εικασία ισχυρίζεται κάτι πολύ απλό: ότι η επανάληψη δυο απλών αριθμητικών πράξεων με μια συγκεκριμένη διαδικασία μπορεί να μετατρέψει οποιονδήποτε θετικό ακέραιο στο 1. Η διαδικασία των πράξεων έχει ως εξής:

1. Διάλεξε οποιονδήποτε θετικό ακέραιο N .
2. Αν ο αριθμός είναι 1, τότε τερμάτισε την διαδικασία.
3. Αν είναι άρτιος, ο επόμενος αριθμός στην ακολουθία θα είναι ο $N/2$.
4. Αν είναι περιττός, ο επόμενος αριθμός στην ακολουθία θα είναι ο $3 \times N + 1$.
5. Επανάλαβε τα βήματα 2-4 μέχρι να φτάσουμε στο 1.

Για παράδειγμα, έστω $N = 3$. Η παραπάνω διαδικασία θα παράξει την ακολουθία: 3, 10, 5, 16, 8, 4, 2, 1. Μπορείτε να δοκιμάσετε το ίδιο με τον αγαπημένο σας θετικό ακέραιο και να ελέγξετε την ακολουθία που παράγεται. Ο δικός μου για παράδειγμα είναι το 42 που οδηγεί στην ακολουθία: 42, 21, 64, 32, 16, 8, 4, 2, 1. Λογικά και η δική σας επιλογή κατέληξε στο 1 μετά από μερικά βήματα - αν όχι ίσως έχετε την ευκαιρία να γίνετε εκατομμυριούχοι (ποιος είπε ότι τα μαθηματικά δεν έχουν λεφτά!).

Για κάθε θετικό ακέραιο N , ο αριθμός στοιχείων της ακολουθίας που παράγεται μέχρι να καταλήξουμε στο 1, λέγεται μήκος ακολουθίας Collatz. Για παράδειγμα, για $N = 3$, το μήκος της ακολουθίας είναι 8 (η ακολουθία έχει 8 στοιχεία: 3, 10, 5, 16, 8, 4, 2, 1). Αντίστοιχα για τον αριθμό 42, το μήκος ακολουθίας Collatz είναι 9. Αν $N = 1$, τότε έχουμε το ελάχιστο μήκος 1.

Σε αυτήν την άσκηση, καλείστε να γράψετε ένα πρόγραμμα collatz.c που να βρίσκει μήκος της ακολουθίας collatz για έναν αριθμό.

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

1.2 Γράψτε σε γλώσσα C μια συνάρτηση `int collatz_it(int n)` που υπολογίζει το μήκος ακολουθίας collatz - δηλαδή τον αριθμό των βημάτων που απαιτούνται προκειμένου ο αριθμός n να φτάσει στο 1. Ολοκληρώστε την υλοποίηση με δομή επανάληψης.

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

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

<pre><Τύπος Επιστροφής> Τύπος δεδομένων της πιμής που επιστρέφεται από τη συνάρτηση μέσω της εντολής: return <παράσταση>;</pre> <p>Σε περίπτωση μη επιστροφής πιμής, ο <Τύπος Επιστροφής> ορίζεται σαν void.</p>	<pre><Τυπικές Παράμετροι> Μεταβλητές, μαζί με τους τύπους τους, που χρησιμοποιούνται στη συνάρτηση, χωρισμένες με κόμμα, στις οποίες δίνονται πιμές από την καλούσα συνάρτηση.</pre>	<p>Προαναγγελία πρωτότυπου συνάρτησης, όταν καλείται πριν ορισθεί</p> <pre><Τύπος Επιστροφής> <Όνομα Συνάρτησης> (<Τυπικές Παράμετροι>); int main(void) { } <Τύπος Επιστροφής> <Όνομα Συνάρτησης> (<Τυπικές Παράμετροι>) { }</pre>
---	---	--

1.3 Υλοποιήστε τον ίδιο αλγόριθμο σε μια συνάρτηση `int collatz(int n)` κάνοντας χρήση αναδρομής και χωρίς να χρησιμοποιήσετε άλλη δομή επανάληψης.

1.4 Ελέγξτε τα αποτελέσματά σας ώστε να βεβαιωθείτε ότι το πρόγραμμά σας λειτουργεί σωστά για διάφορες αρχικές τιμές:

```
$ ./collatz
Number to find the length of the collatz sequence: 42
Iterative result: 9
Recursive result: 9
$ ./collatz
Number to find the length of the collatz sequence: 950000001
Iterative result: 199
Recursive result: 199
```

Κάποιο από τα αποτελέσματά μας δεν συμφωνεί; Τι μπορεί να πηγαίνει στραβά;

1.5 (Advanced - Προαιρετικό) Κάνετε το πρόγραμμά σας να δέχεται τον ακέραιο από την γραμμή εντολών, για παράδειγμα:

```
$ ./collatz 950000001
Iterative result: 199
Recursive result: 199
```

Άσκηση 2. Η ακολουθία Fibonacci (fib.c).

Η ακολουθία Fibonacci είναι από τις πιο διάσημες μαθηματικές ακολουθίες. Ο αναδρομικός ορισμός της συνάρτησης είναι ιδιαίτερα απλός:

$$fib(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ fib(n - 1) + fib(n - 2) & \text{if } n \geq 2. \end{cases}$$

και προκύπτει η ακολουθία (συνδέεται και με την [χρυσή τομή](#)):

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Σε αυτήν την άσκηση θα υλοποιήσετε ένα πρόγραμμα `fib.c` που υπολογίζει τον n -οστό αριθμό της ακολουθίας Fibonacci.

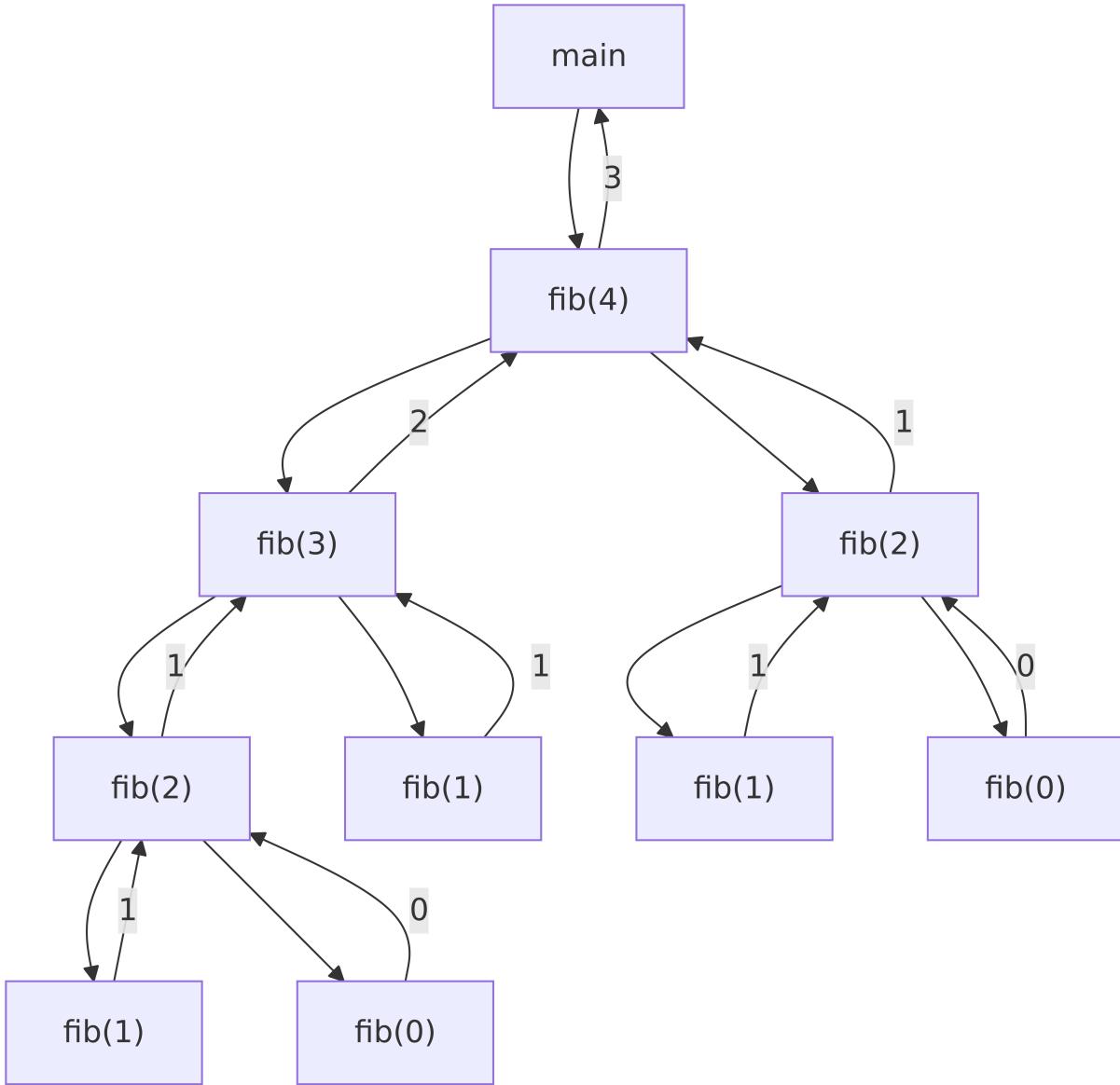
2.1: Ορίστε την αναδρομική συνάρτηση `int fib(int n)` που να υλοποιεί τον υπολογισμό του n -οστού όρου της ακολουθίας Fibonacci.

2.2 Μετατρέψτε το πρόγραμμά σας έτσι ώστε να δέχεται από τον χρήστη το [πόσους αριθμούς](#) Fibonacci θέλει να εκτυπωθούν [και στην συνέχεια](#) θα καλεί την συνάρτηση `fib` για να εκτυπώσει τον κάθε όρο. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./fib
How many fibonacci terms would you like: 35
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
...
fib(31) = 1346269
fib(32) = 2178309
fib(33) = 3524578
fib(34) = 5702887
```

Τρέξτε το πρόγραμμά σας για περισσότερους όρους, για παράδειγμα 45. Τι παρατηρείτε;

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. Γράψτε μια συνάρτηση `int fib_it(int n)` η οποία να υπολογίζει τον n -οστό όρο της ακολουθίας Fibonacci ακολουθώντας την παραπάνω επαναληπτική διαδικασία. Μετατρέψτε το πρόγραμμά σας ώστε να χρησιμοποιεί την επαναληπτική διαδικασία και ξανατρέξτε το παραπάνω πείραμα. Τι παρατηρείτε;

2.6 (Advanced - Προαιρετικό) Μπορείτε να υπολογίσετε τον n -οστό όρο Fibonacci αναδρομικά

με απόδοση κοντά στον επαναληπτικό αλγόριθμο; Αν ναι, υλοποιήστε την λύση σας σε μια συνάρτηση `fib_rec_fast` και μετατρέψτε το πρόγραμμά σας ώστε να χρησιμοποιεί την συνάρτηση `fib_rec_fast`.

Άσκηση 3 (Παλιό Θέμα / Προαιρετικό). Σκαλί-Σκαλί (ladder.c)

Ένα παιδί ανεβαίνει μια σκάλα και σε κάθε βήμα του ανεβαίνει 1, 2 ή 3 σκαλιά. Έστω ότι η σκάλα έχει 5 σκαλιά. Ένας τρόπος να τα ανέβει είναι 1-1-1-1-1, δηλαδή ένα σκαλί την φορά. Ένας άλλος είναι ο 2-2-1 (δύο σκαλιά στα πρώτα δύο βήματα και μετά ένα σκαλί). Άλλοι πιθανοί τρόποι είναι οι 3-2, 2-3, 3-1-1, κ.ο.κ - σε κάθε περίπτωση ο συνολικός αριθμός των σκαλιών που ανέβηκε πρέπει να ισούται με τον αριθμό των σκαλιών της σκάλας. Γράψτε ένα πρόγραμμα `ladder.c` το οποίο διαβάζει τον συνολικό αριθμό των σκαλοπατιών και επιστρέφει τον αριθμό των διαφορετικών τρόπων με τους οποίους το παιδί μπορεί να ανέβει την σκάλα. Το πρόγραμμά σας θέλουμε να βγάζει σωστά αποτελέσματα για σκάλες μέχρι 50 σκαλιά. Λάβετε υπόψη σας ότι στην C δεν μπορούν να αναπαρασταθούν ακέραιοι μεγαλύτεροι από το $2^{64} - 1$, δεδομένου ότι ο ευρύτερος τύπος ακεραίου που μπορούμε να έχουμε είναι ο `unsigned long long` (των 8 bytes). Ακολουθούν ενδεικτικές εκτελέσεις:

```
$ ./step
Please provide the number of steps: 4
There are 7 different ways to climb the ladder
$ ./step
Please provide the number of steps: 5
There are 13 different ways to climb the ladder
$ ./step
Please provide the number of steps: 6
There are 24 different ways to climb the ladder
$ ./step
Please provide the number of steps: 50
There are 10562230626642 different ways to climb the ladder
```

Εργαστήριο #6: Δείκτες και Πίνακες

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 αλγορίθμικές 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 - myprog.c

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.

Υπενθυμίζουμε ότι 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 - sieve.c

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 αλγορίθμους 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.

Όταν ξεκινά ο αλγόριθμος, όλοι οι αριθμοί θεωρούνται πιθανοί πρώτοι (αρχικοποιημένοι στο 1):

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Ο «2» είναι πρώτος, διαγραφή των 4, 6, 8, 10, 12, 14, 16, 18, 20 (το σημειώνουμε με *):

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Ο «3» είναι πρώτος, διαγραφή των 6, 9, 12, 15, 18:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0

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 "i is a prime number"

Εκτελέστε το πρόγραμμά σας και επιβεβαιώστε την ορθότητα του αποτελέσματος.

Άσκηση 3: Πίνακες και αριθμητική δεικτών - pointers.c

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

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 *array, 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 *array, 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 *array, 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 τυπώνει τη βαθμολογία με ακρίβεια 2 δεκαδικών ψηφίων σύμφωνα με τον τύπο:

Βαθμός = (Άθροισμα - Μέγιστο - Ελαχιστο) / 8

Έστω ένα αρχείο grades.txt με τις παρακάτω βαθμολογίες:

8 2 3 9 0 10 8 3 4 5

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./judgement < grades.txt
```

```
Sum: 52
```

```
Min: 0
```

```
Max: 10
```

```
Average: 5.25
```

Εργαστήριο #7: 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 - (twodim.c)

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

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

Φτιάξτε μια συνάρτηση `print_array` που να εκτυπώνει τον πίνακα κατά γραμμές, με τα στοιχεία κάθε γραμμής χωρισμένα με στηλογνώμονα (tab).

1.2 Επεκτείνετε το πρόγραμμα `twodim.c` με μια συνάρτηση `print_transposed` ώστε να εκτυπώνει και τον ανάστροφο του πίνακα, δηλαδή αυτόν που έχει στήλες τις γραμμές του αρχικού και γραμμές τις στήλες του αρχικού (αυτή είναι μια κατεξοχήν άσκηση προγραμματιστικής συνέντευξης).

Επομένως, αν ένας αρχικός πίνακας 3 x 5 που εκτυπώνεται έτσι:

```
1 4 6 7 8
5 6 7 1 3
1 0 4 7 2
```

όταν τον τυπώνετε ανάστροφα, θέλουμε να τυπώνεται 5 x 3, δηλαδή ως εξής:

```
1 5 1
4 6 0
6 7 4
7 1 7
8 3 2
```

1.3 Επεκτείνετε το πρόγραμμα `twodim.c` με μια συνάρτηση `print_reverse` ώστε να εκτυπώνει τις γραμμές του αρχικού πίνακα με αντίστροφη σειρά. Για παράδειγμα, αν ο αρχικός σας πίνακας είναι και πάλι:

```
1 4 6 7 8
5 6 7 1 3
1 0 4 7 2
```

όταν τυπωθεί με αντίστροφη σειρά, θέλουμε να τυπώσει κάτι τέτοιο:

```
8 7 6 4 1  
3 1 7 6 5  
2 7 4 0 1
```

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

Ο αρχικός πίνακας:

```
1 4 6 7 8  
5 6 7 1 3  
1 0 4 7 2
```

Θέλουμε να εκτυπωθεί ως εξής:

```
1 4 6 7 8 3 1 7 6 5 1 0 4 7 2
```

Άσκηση 2: Δυναμική δέσμευση μνήμης για μονοδιάστατο πίνακα - (array.c)

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; // Δήλωση ενός δείκτη σε στοιχεία τύπου TΔ`

`p = malloc(N * sizeof(TΔ)); // Δέσμευση μνήμης για N στοιχεία τύπου 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`.

Γράψτε μια συνάρτηση `print_array` που να εκτυπώνει 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, με μια συνάρτηση `print_average` ώστε να εκτυπώνεται ο μέσος όρος των στοιχείων του πίνακα.

Άσκηση 3: Δυναμική δέσμευση μνήμης για διδιάστατο πίνακα - (mines.c)

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.

Στην συνέχεια παραθέτουμε τρόπους για να δεσμεύσουμε και να αποδεσμεύσουμε μνήμη δυναμικά για δισδιάστατους πίνακες. Αντίστοιχα μπορείτε να διαχειριστείτε πίνακες με περισσότερες διαστάσεις. Στην τελική εξέταση θα χρειαστεί να διαχειριστείτε δεδομένα δυναμικής φύσης (δηλαδή δεδομένα που πρέπει να χωρέσουν σε πίνακες των οποίων τα μεγέθη δεν γνωρίζουμε εκ των προτέρων) και επομένως είναι καλό να εξοικειωθείτε με την διαδικασία δέσμευσης/αποδέσμευσης μνήμης.

Δυναμική δέσμευση μνήμης για δισδιάστατο πίνακα διάστασης $N \times M$

```
TΔ **p;
p = malloc(N * sizeof(TΔ *));
if (p == NULL) {
    fprintf(stderr, "Failed to allocate rows\n");
    exit(1);
}
for (i = 0 ; i < N ; i++) {
    p[i] = malloc(M * sizeof(TΔ));
    if (p[i] == NULL) {
        fprintf(stderr, "Failed to allocate column %d\n", i);
        exit(1);
    }
}
```

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

```
int i;
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 . óταν δεν υπάρχει νάρκη και ο χαρακτήρας * óταν υπάρχει νάρκη.

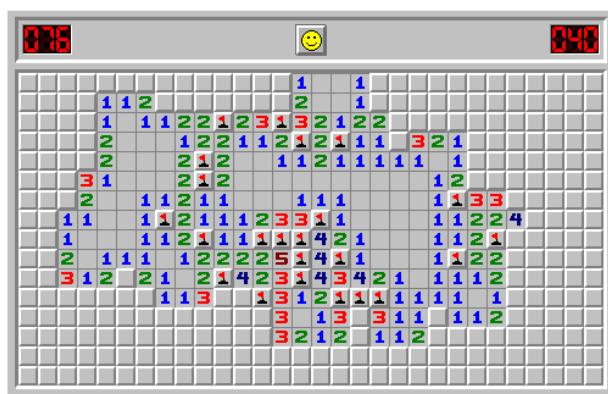
```
3 4  
..*.  
.**.  
*.*.
```

3.3 Επεκτείνετε το πρόγραμμά σας, ώστε να εκτυπώνονται τα περιεχόμενα του πίνακα που διαβάστηκε. Εκτελέστε το πρόγραμμα 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
```

Μπορείτε να δοκιμάσετε το πρόγραμμά σας και με μεγαλύτερα πλέγματα προκειμένου να επιβεβαιώσετε ότι λειτουργεί σωστά.



ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 3η) - (my_prog.c)

Σε προηγούμενα εργαστήρια 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 προσπελαύνουμε 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(int argc, char *argv[]) {
    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(int argc, char *argv[]) {
    int array[10];
    array[10] = 5;
    return 0;
}
```

Εδώ, πηγαίνουμε και προσπελαύνουμε 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. Για αυτό το λόγο, αλλά και για την ευκολότερη ανίχνευση και των λογικών λαθών στα οποία αναφερθήκαμε σε προηγούμενα εργαστήρια, θα δούμε στη συνέχεια πώς δουλεύει ένα εργαλείο αποσφαλμάτωσης (debugging), o 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 συντετμημένα 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 μπορέσουμε να εκτελέσουμε βηματικά τον κώδικα μας.

run όρισμα1 όρισμα2 ... όρισμαn (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(int argc, char *argv[]) {
    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 ρ 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.

```
(gdb) p p[i]
```

```
Cannot access memory at address 0xbff8de000
```

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

```
(gdb) p i
```

```
$3 = 1279
```

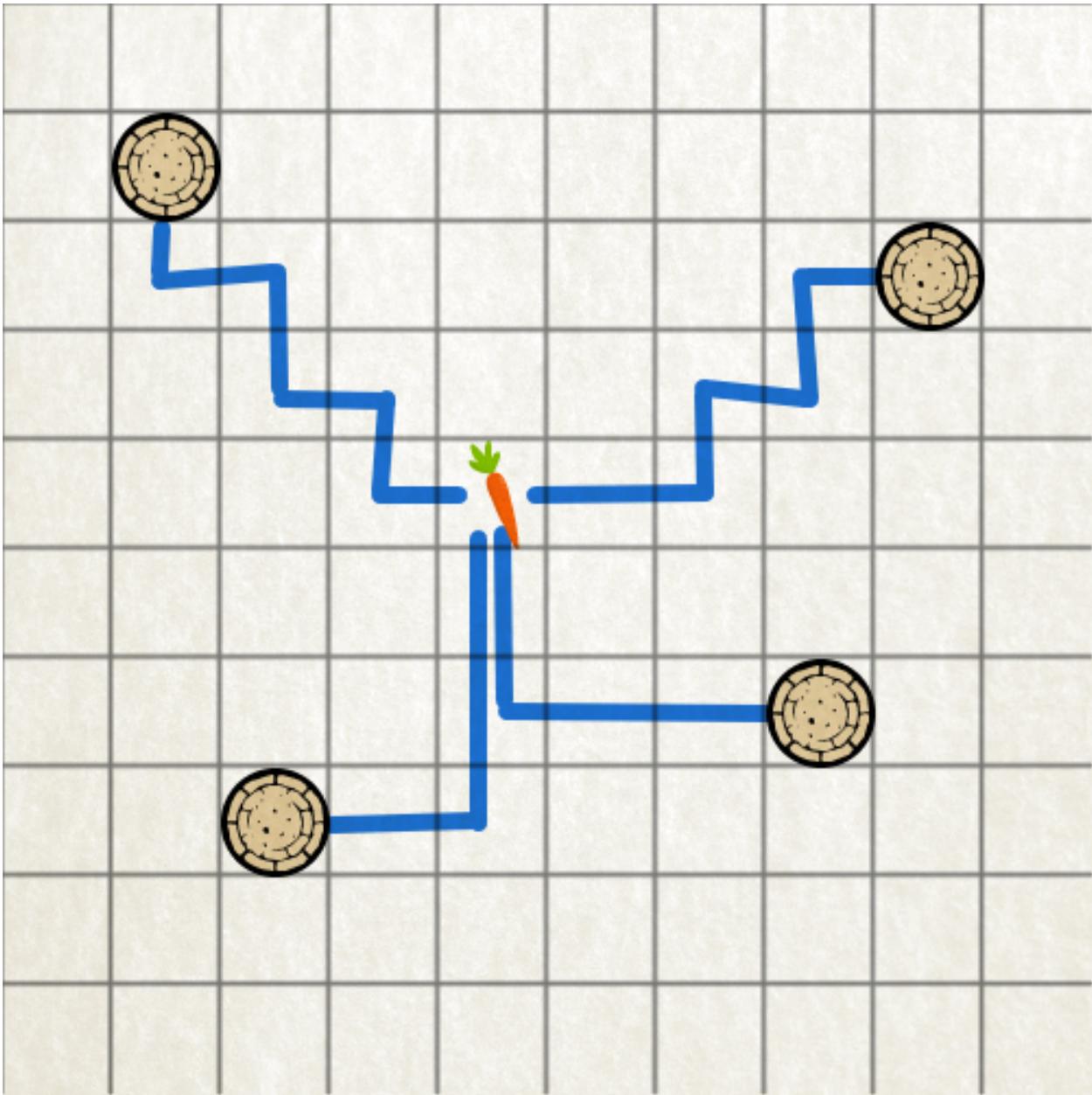
Το `$3` δεν πρέπει να σας απασχολεί. Αυτό που μας ενδιαφέρει είναι η αριθμητική τιμή που εμφανίζεται, το 1279. Άρα το `i` έχει φτάσει στο 1279! Είναι προφανές ότι κάπου μέσα στο πρόγραμμά μας έχουμε ξεχάσει να κάνουμε έλεγχο για να μην ξεπερνάμε τα όρια του πίνακα `p`. Όντως, πουθενά στον κώδικά μας 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`.

Άσκηση 5 (Παλιό Θέμα): Χτίζοντας έναν Χιονάνθρωπο - (olaf.c)

Βρισκόμαστε σε έναν χιονισμένο τετράγωνο κήπο διαστάσεων NxN και Θέλουμε να φτιάξουμε έναν χιονάνθρωπο. Δυστυχώς το χιόνι είναι μαζεμένο σε συγκεκριμένους σωρούς μέσα στον κήπο (δεν είναι παντού). Γράψτε ένα πρόγραμμα που αποφασίζει που πρέπει να φτιάξουμε τον χιονάνθρωπό μας ώστε να κάνουμε τον ελάχιστο δυνατό κόπο. Για λόγους απλοποίησης θεωρούμε ότι όλες οι συντεταγμένες στον κήπο είναι ακέραιοι και ότι οι κινήσεις μας μπορούν να είναι μόνο πάνω-κάτω-αριστερά-δεξιά (όχι διαγώνια). Το μέγεθος του κήπου και οι τοποθεσίες των σωρών χιονιού δίνονται στο πρόγραμμα μέσω αρχείου του οποίου το όνομα θα είναι το πρώτο όρισμα του προγράμματος. Το αρχείο θα περιέχει την διάσταση του κήπου (N) ακολουθούμενη από τις συντεταγμένες του κάθε σωρού. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o olaf olaf.c
$ cat map.txt
10
1 1
6 7
2 8
7 2
$ ./olaf map.txt
We will position the snowman on (4, 4) with a minimum cost of 22.
```

Παρακάτω βλέπουμε την οπτικοποίηση της επιλογής μας - ο χιονάνθρωπος βρίσκεται στην θέση "καρότο" ενώ οι σωροί με τα χιόνια απεικονίζονται με τους κύκλους. Από τους δύο πάνω σωρούς παρατηρούμε ότι χρειαζόμαστε 6 βήματα για να φτάσουμε τον χιονάνθρωπο ενώ από τους δύο κάτω χρειαζόμαστε 5 - σύνολο ($6 + 6 + 5 + 5 =$) 22. Παρατηρήστε ότι μπορεί να υπάρχουν πάνω από μία βέλτιστες τοποθετήσεις, χρειάζεται να βρούμε μόνο μία από αυτές. Για λόγους απλότητας ο χιονάνθρωπος μπορεί να τοποθετηθεί πάνω σε έναν σωρό (και τα βήματα που απαιτούνται σε αυτήν την περίπτωση είναι 0).



Άσκηση 6 (Παλιό Θέμα): Κινήσεις σε Πλέγμα - (pacman.c)

Γράψτε ένα πρόγραμμα που παίρνει δύο ορίσματα: (1) το όνομα ενός αρχείου που περιέχει μια πίστα pacman και (2) τις κινήσεις που πρέπει να κάνει ο pacman πάνω στο πλέγμα και τυπώνει την τελική κατάσταση του παιχνιδιού μετά από αυτές τις κινήσεις. Το αρχείο θα έχει την ακόλουθη μορφή:

1. Διάσταση του πλέγματος του παιχνιδιού (δεκαδικός ακέραιος).
2. Τετράγωνο πλέγμα με χαρακτήρες '.' και ένα 'P' για την αρχική θέση του pacman. Όλοι οι άλλοι χαρακτήρες αγνοούνται.

Οι κινήσεις που πρέπει να κάνει ο pacman θα αποτελούνται από 4 χαρακτήρες 'U' (Up), 'D' (Down), 'L' (Left), 'R' (Right). Όταν ο pacman κινείται σε κάποια θέση, τότε η τελεία ('.') σε εκείνη την θέση εξαφανίζεται. Η πίστα είναι τόρος, δηλαδή αν είσαι στην τελευταία γραμμή της και κινηθείς προς τα κάτω (D) τότε μεταφέρεσαι στην πρώτη γραμμή. Αντίστοιχα για κινήσεις αριστερά-δεξιά. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o pacman pacman.c
$ cat level.txt
7
.....
.....
..P....
.....
.....
.....
.....
$ ./pacman level.txt RRD
.....
.....
..
..
....P..
.....
.....
.....
$ ./pacman level.txt RRDDLL
.....
.....
..
..
....P..
.....
.....
$ ./pacman level.txt RRDDLLUU
.....
.....
..P ..
...
...
.....
.....
$ ./pacman level.txt RRDDLLUULLLL
.....
.....
```

P

... . . .
.. ..
.....
.....

Εργαστήριο 8: Συμβολοσειρές και Ορίσματα Γραμμής Εντολών

Στο εργαστήριο αυτό θα δούμε πώς ορίζονται και πώς χρησιμοποιούνται οι συμβολοσειρές (strings) 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 (command line arguments).

Άσκηση 1: Επεξεργασία συμβολοσειρών - (string.c)

1.1 Υλοποιήστε τη συνάρτηση `int mystrlen(char *str)` 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`.

1.3 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.

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

1.3.2 Αντιγράψτε στην `strA` τη συμβολοσειρά "This is a string." και στην `strB` τη συμβολοσειρά "This is another string." χρησιμοποιώντας την συνάρτηση `strcpy`.

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

Η συνάρτηση `strcpy` αντιγράφει τη συμβολοσειρά `s2` στην `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

1.3.3 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)
```

Η συνάρτηση `strlen` επιστρέφει στο όνομα της το μήκος της συμβολοσειράς `s` (χωρίς να μετράται το τελικό '\0').

1.3.4 Συγκρίνετε αλφαριθμητικά τις συμβολοσειρές `strA` και `strB` εκτυπώνοντας κατάλληλο μήνυμα.

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

Η συνάρτηση `strcmp` συγκρίνει τις συμβολοσειρές `s1` και `s2` χαρακτήρα προς χαρακτήρα με βάση τους αντίστοιχους ASCII κωδικούς. Επιστρέφει:

1. = 0, αν οι συμβολοσειρές είναι ίδιες
2. > 0, αν η `s1` είναι λεξικογραφικά «μεγαλύτερη» της `s2`
3. < 0, αν η `s1` είναι λεξικογραφικά «μικρότερη» της `s2`

1.3.5 Προσαρτήστε τη συμβολοσειρά `strB` στο τέλος της `strA` (χρησιμοποιώντας τη συνάρτηση `mystrcat` που υλοποιήσατε παραπάνω) 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)
```

Η `strcat` προσαρτά / συνενώνει (concatenates) ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

1.3.6 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. Ένα παράδειγμα χρήσης ακολουθεί με ένα `quote`:

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

Προσθέστε το παραπάνω σε μια συνάρτηση `strtok_example` και τρέξτε την από την `main`. Στο `stdout` θα πρέπει να δείτε μια ακολουθία της μορφής:

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

far

Άσκηση 2: Ορίσματα γραμμής εντολής - (calc.c)

2.1 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 ακολουθούν:

```
$ ./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[0], ..., 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.

Άσκηση 3 (Παλιό θέμα): Πετυχαίνοντας τον στόχο - (legolas.c)

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως ορίσματα από την γραμμή εντολών έναν ακέραιο-στόχο (goal το argv[1]) και στην συνέχεια ένα σύνολο υποψηφίων ακεραίων (candidates) και τυπώνει όλους τους συνδυασμούς 3 υποψηφίων των οποίων το άθροισμα ισούται με τον στόχο. Η σειρά με την οποία εκτυπώνονται τα αποτελέσματα δεν έχει σημασία για την ορθότητα του προγράμματος. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o legolas legolas.c  
$ ./legolas 42 19 21 3 5 12  
No combination of candidates leads to 42  
$ ./legolas 42 19 21 3 5 12 11  
Candidates combination found: 19 + 12 + 11 = 42  
$ ./legolas 42 27 25 12 31 5 26 40 34 3 18  
Candidates combination found: 27 + 12 + 3 = 42  
$ ./legolas 25 12 5 34 3 18  
Candidates combination found: 25 + 12 + 5 = 42  
$ ./legolas 5 34 3 18  
Candidates combination found: 5 + 34 + 3 = 42  
$ ./legolas 37372082074 9238742398 82934723 27893492387 127863435 239847289  
Candidates combination found: 9238742398 + 27893492387 + 239847289 = 37372082074
```

ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 4η)

Στο εργαστήριο 8 είδαμε ένα εργαλείο για την αποσφαλμάτωση προγραμμάτων, τον debugger gdb. Στο σημερινό εργαστήριο, θα χρησιμοποιήσουμε και πάλι debuggers 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>

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(int argc, char ** argv) {

    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. Μπορείτε να βρείτε το λάθος μέσω gdb;

Αν προσπαθήσουμε λίγο, θα 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 4: Υπολογισμός Μισθών - (wages.c)

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 και διορθώστε το.

```
#include <stdio.h>

#define N 100

int main(int argc, char ** argv) {
    int i = 0, sum = 0;
    int *wages = NULL;
    srand(N);
    for (i = 0 ; i < N ; i++)
        wages[i] = 700 + (rand()%301) - 1000;
    while (wages[i] > 0 && i < N) {
        sum += wages[i];
        i++;
    }
    printf("Average wage is %.2f\n", sum/N);
    return 0;
}
```

Εργαστήριο 9: Δομές και Αυτοαναφορικές Δομές

Σε αυτό το εργαστήριο θα μελετήσουμε τις δυνατότητες που μας προσφέρει η C για να ομαδοποιούμε δεδομένα χρησιμοποιώντας δομές. Μέσω των δομών θα κατασκευάσουμε συνδεδεμένες λίστες και δυαδικά δέντρα, τα οποία ονομάζονται αυτοαναφορικές δομές.

Άσκηση 1: Δομές και συναρτήσεις - (point.c)

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)
```

Η συνάρτηση θα υπολογίζει και θα επιστρέψει το σημείο που βρίσκεται στο μέσο του ευθυγράμμου τμήματος με άκρα τα σημεία a και b.

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

Άσκηση 2: Δομές και δείκτες - (person.c)

2.1 Δημιουργήστε το αρχείο person.c και ορίστε τη δομή person που αποθηκεύει το όνομα, το επώνυμο και το πατρώνυμο ενός ατόμου.

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

2.2 Κατασκευάστε τη συνάρτηση:

```
struct person *person_init(char *firstname, char *lastname, char *middlename);
```

Η συνάρτηση θα δεσμεύει χώρο για μία δομή τύπου person, θα την αρχικοποιεί και θα επιστρέψει τη διεύθυνσή της. Καλέστε τη συνάρτηση από την main για να καταχωρίσετε τα στοιχεία του πατέρα σας.

2.3 Ορίστε τη συνάρτηση:

```
struct person *childof(struct person father, char *newname);
```

Η συνάρτηση θα καταχωρεί τα στοιχεία ενός παιδιού με μικρό όνομα newname, χρησιμοποιώντας τον πατέρα του father. Καλέστε τη συνάρτηση από την main για να καταχωρίσετε τα στοιχεία σας.

Άσκηση 3: Συνδεδεμένες λίστες - (grades.c)

3.1 Κατασκευάστε το πρόγραμμα grades.c και ορίστε μία αυτοαναφορική δομή λίστας ακεραίων αριθμών.

```
typedef struct listnode *Listptr;

struct listnode {
    int data;
    Listptr next;
};
```

3.2 Κατασκευάστε τη συνάρτηση:

```
void insert_at_start(Listptr *ptr, int grade);
```

Η συνάρτηση θα προσθέτει έναν βαθμό στην αρχή της λίστας. Τροποποιήστε τη main για να διαβάζει βαθμούς από το πληκτρολόγιο και να τους προσθέτει στη λίστα.

3.3 Κατασκευάστε τη συνάρτηση:

```
float average(Listptr ptr);
```

Η συνάρτηση θα διασχίζει τα περιεχόμενα της λίστας και θα υπολογίζει τον μέσο όρο των βαθμών που είναι αποθηκευμένοι σε αυτήν.

Άσκηση 4: Δυαδικά δένδρα - (tree.c)

4.1 Δημιουργήστε το αρχείο tree.c και ορίστε μία αυτοαναφορική δομή δυαδικού δένδρου ακεραίων αριθμών.

```
typedef struct tnode *Treeptr;

struct tnode {
    int data;
    Treeptr left;
    Treeptr 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. Η ιδιότητα αυτή ισχύει τόσο για το αριστερό όσο και για το δεξί υποδένδρο.

Ορίστε την αναδρομική συνάρτηση:

```
Treeptr addtree(Treeptr p, int x)
```

Η συνάρτηση προσθέτει έναν αριθμό x στο ταξινομημένο δυαδικό δένδρο p , διατηρώντας το ταξινομημένο, και επιστρέφει το νέο δένδρο. Ο αλγόριθμος λειτουργεί ως εξής:

- Αν το δένδρο p είναι κενό (NULL), δημιουργείται ένας νέος κόμβος που περιέχει τον αριθμό x .
- Αν ο αριθμός που περιέχει ο τρέχων κόμβος είναι μεγαλύτερος από το x , η συνάρτηση καλείται για το αριστερό παιδί του κόμβου.
- Αν ο αριθμός είναι μικρότερος από το x , η συνάρτηση καλείται για το δεξί παιδί.
- Αν ο αριθμός είναι ίσος με το x , δεν γίνεται εισαγωγή.

Τροποποιήστε τη συνάρτηση `main` 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(Treeptr p);
```

Η συνάρτηση δέχεται ως όρισμα το δένδρο και εκτυπώνει τα περιεχόμενά του με **in-order** διάσχιση. Ο αλγόριθμος λειτουργεί ως εξής:

1. Αν το δένδρο είναι κενό (NULL), η συνάρτηση επιστρέφει.
2. Διαφορετικά, εκτελούνται τα εξής βήματα:
 - Καλείται η `treeprint` για το αριστερό παιδί.
 - Εκτυπώνεται η τιμή του τρέχοντος κόμβου.
 - Καλείται η `treeprint` για το δεξί παιδί.

Καλέστε τη `treeprint` από τη συνάρτηση `main` για να εκτυπώσετε το δένδρο που κατασκευάσατε.

Εργαστήριο 10: Είσοδος και Έξοδος με 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 - (more.c)

Κατασκευάστε το πρόγραμμα more.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.

Συναρτήσεις Εισόδου/Εξόδου στην C

Ανοιγμα και Κλείσιμο Αρχείων

```
FILE *fopen(const char *filename, const char *mode);  
int fclose(FILE *fp);
```

- fopen: Ανοίγει το αρχείο με όνομα filename με τρόπο προσπέλασης που καθορίζεται από το mode. Επιστρέφει ένα ρεύμα μέσω του οποίου μπορούμε στη συνέχεια να αναφερόμαστε στο αρχείο, ή NULL αν δεν ήταν δυνατόν να ανοίξει το αρχείο.
 - mode Παραδείγματα:
 - * "r": Διάβασμα από υπάρχον αρχείο.
 - * "w": Γράψιμο σε αρχείο. Δημιουργείται αν δεν υπάρχει ή διαγράφονται τα περιεχόμενα αν υπάρχει.
 - * "a": Προσάρτηση δεδομένων στο τέλος του αρχείου.
 - * "r+", "w+", "a+": Διάφοροι συνδυασμοί ανάγνωσης και εγγραφής.
- fclose: Κλείνει το ρεύμα fp. Επιστρέφει 0 σε περίπτωση επιτυχίας.

Άλλες Συναρτήσεις

Συνάρτηση	Περιγραφή
int feof(FILE *fp)	Επιστρέφει τιμή διαφορετική από το 0 αν το προηγούμενο διαβασμό ήταν το EOF.
int fprintf(FILE *fp, ...)	Αντίστοιχη της printf, αλλά γράφει στο ρεύμα fp αντί του παραγόμενου ρεύματος.
int fscanf(FILE *fp, ...)	Αντίστοιχη της scanf, αλλά διαβάζει από το ρεύμα fp αντί της παραγόμενης σειράς.
char *fgets(char *buf, int max, FILE *fp)	Διαβάζει το πολύ max-1 χαρακτήρες από το ρεύμα fp μέχρι την επόμενη μετατόπιση.
int getc(FILE *fp)	Επιστρέφει τον επόμενο χαρακτήρα από το ρεύμα fp, ή EOF αν δεν υπάρχει.

Άσκηση 2: Δυαδικά αρχεία - (grades.c)

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.

Συναρτήσεις για Δυαδική Εισαγωγή/Εξαγωγή:

Συνάρτηση	Περιγραφή
<code>size_t fread(void *ptr, size_t size, size_t count, FILE *fp)</code>	Διαβάζει από το ρεύμα <code>fp</code> , το
<code>size_t fwrite(const void *ptr, size_t size, size_t count, FILE *fp)</code>	Γράφει στο ρεύμα <code>fp</code> το πολύ

Για να δείτε τα περιεχόμενα του αρχείου `grades.dat`, αν δουλεύετε σ' ένα Unix σύστημα, χρησιμοποιήστε την εντολή `"od -tu1c grades.dat"`.

Άσκηση 3: Σύγκριση Αρχείων - (filediff.c)

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: Μέτρηση Στατιστικών Αρχείων - (count.c)

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.