Προλεγόμενα

Το βιβλίο αυτό αποτελεί εξέλιξη των σημειώσεων του μαθήματος «Εισαγωγή στον προγραμματισμό», το οποίο διδάσκω στο Πανεπιστήμιο Ιωαννίνων από το 2014. Στόχος του είναι να εισαγάγει τον αναγνώστη σε βασικές προγραμματιστικές έννοιες και βασικά εργαλεία, που θα τον βοηθήσουν να κάνει τα πρώτα του βήματα στον προγραμματισμό. Μελετώντας το βιβλίο, εφαρμόζοντας τα παραδείγματα που δίνονται και λύνοντας τις ασκήσεις στο τέλος κάθε κεφαλαίου, ο επίδοξος προγραμματιστής εξοικειώνεται με τα σχετικά εργαλεία και καλλιεργεί την αλγοριθμική σκέψη μαθαίνοντας να λύνει μαθηματικά και λογιστικά προβλήματα.

Η γλώσσα προγραμματισμού που επιλέχθηκε για την εκμάθηση προγραμματισμού είναι η Python. Αυτό έγινε λόγω της πολύ μεγάλης δημοφιλίας της συγκεκριμένης γλώσσας, αλλά και λόγω της ευκολίας που παρέχει στον προγραμματιστή όσον αφορά την εγκατάστασή της, καθώς και την ανάπτυξη, εκτέλεση και αποσφαλμάτωση προγραμμάτων.

Όπως λέω πάντα στους φοιτητές μου, ο προγραμματισμός δεν μαθαίνεται απλώς διαβάζοντας σημειώσεις ή ένα βιβλίο. Χρειαζόμαστε οπωσδήποτε εξάσκηση: αν δεν γράψεις προγράμματα, δεν μαθαίνεις να προγραμματίζεις. Άρα περιμένω από τους φοιτητές μου να διαθέτουν τα εξής:

- **Έναν υπολογιστή** με εγκατεστημένη τη γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε για την εκμάθηση. Ο υπολογιστής δεν χρειάζεται να είναι καινούργιος ή να έχει υψηλές

αποδόσεις. Ακόμα και με έναν υπολογιστή εικοσαετίας μπορείτε άνετα να μάθετε να προγραμματίζετε. Στην παρούσα φάση, η γλώσσα που απαιτείται για το μάθημα είναι η Python (έκδοση 3). Θα πρέπει επίσης να έχετε εγκατεστημένο ένα ολοκληρωμένο περιβάλλον ανάπτυξης προγραμμάτων (π.χ. το IDLE της Python) ή έναν κειμενογράφο που να αναγνωρίζει κώδικα και να χρωματίζει αυτόματα δεσμευμένες λέξεις της γλώσσας, σχόλια κ.λπ. Ανάλογα με το λειτουργικό σύστημα με το οποίο εργάζεστε, υπάρχουν δωρεάν κειμενογράφοι με αυτή τη λειτουργικότητα (π.χ. ο Notepad++ στα Windows, ο gedit στο Linux, ο Sublime σε Mac OS κ.λπ.). Θα πρέπει να μάθετε να χρησιμοποιείτε το IDLE ή τον κειμενογράφο, ώστε να γράφετε τα προγράμματά σας, να διαχειρίζεστε τα αρχεία στα οποία περιέχονται, να εκτελείτε τα προγράμματα, να τα αποσφαλματώνετε κ.λπ.

- **Χρόνο** τον οποίο θα αφιερώνετε τακτικά για την εκμάθηση. Ο χρόνος αυτός θα πρέπει να είναι αρκετός (π.χ. δύο ώρες την ημέρα) και *συνεχής*. Για όσο χρόνο ασχολείται κάποιος με την εκμάθηση προγραμματισμού, πρέπει να είναι απερίσπαστος ώστε να παραμένει συγκεντρωμένος. Αυτό σημαίνει ότι κατά τη διάρκεια της εργασίας μας αφήνουμε το κινητό μας, δεν μπαίνουμε στα social media, δεν απαντάμε σε μηνύματα και γενικά ενημερώνουμε τα άτομα του περιβάλλοντός μας ότι δεν θα είμαστε διαθέσιμοι.

Επιπλέον, χρειάζονται βασικές γνώσεις χειρισμού ενός υπολογιστή, δηλαδή πώς δημιουργούμε και γράφουμε αρχεία κειμένου, πώς σώζουμε και αντιγράφουμε αρχεία στον υπολογιστή, πώς οργανώνουμε αρχεία στον δίσκο, καθώς και πληκτρολόγηση, εκτέλεση προγραμμάτων από παραθυρικό περιβάλλον ή από τη γραμμή εντολών κ.λπ. Τέλος, απαιτείται καλό επίπεδο αγγλικών, αφού η μεγαλύτερη βοήθεια που παίρνουμε όταν μαθαίνουμε προγραμματισμό προέρχεται από πόρους στο διαδίκτυο (π.χ. συζητήσεις, βιβλία και σημειώσεις, παραδείγματα κ.λπ.), οι οποίοι είναι κατά κανόνα γραμμένοι στα αγγλικά.

Στο τέλος κάθε κεφαλαίου υπάρχουν ασκήσεις, τις οποίες συνιστώ να λύσετε. Κάποιες ενότητες και ασκήσεις σημειώνονται με αστερίσκο. Είναι πιο δύσκολες από τις υπόλοιπες και πιθανώς όχι απαραίτητες για τη συνέχεια του βιβλίου, πράγμα που σημαίνει ότι μπορείτε να τις παραλείψετε όταν το διαβάζετε για πρώτη φορά και να επανέλθετε σε αυτές όταν αποκτήσετε αρκετές γνώσεις και εμπειρία.

Το υλικό του πρώτου κεφαλαίου βασίζεται σε ιδέες του Chui Chun Kit από το Πανεπιστήμιο του Χονγκ Κονγκ. Κάποιο υλικό και ορισμένα παραδείγματα προέρχονται από διαφάνειες προηγούμενων διδασκόντων του μαθήματος «Εισαγωγή στον προγραμματισμό» στο Πανεπιστήμιο Ιωαννίνων, δηλαδή των συναδέλφων Γ. Μανή, Λ. Κόντη και Π. Τσαπάρα. Θα ήθελα να ευχαριστήσω τους

Μαρία Χρόνη, Χρυσάνθη Κοσυφάκη, Ντίνο Λαμπρόπουλο, Δημήτρη Τσιτσίγκο και Γιώργο Χριστοδούλου για τη βοήθεια που προσέφεραν στο μάθημα, καθώς και για τον χρόνο που αφιέρωσαν για να διαβάσουν την πρώτη έκδοση των σημειώσεών μου, οι οποίες εξελίχθηκαν στο βιβλίο αυτό. Τους ευχαριστώ επίσης για τις παρατηρήσεις που έκαναν σχετικά με το υλικό και την παρουσίαση των σημειώσεων και για τα σφάλματα που υπέδειξαν.

Μάιος 2023
Ν. Μ.

Κεφάλαιο 1

Εισαγωγή στον προγραμματισμό

Προγραμματισμός ονομάζεται ο σχεδιασμός και η ανάπτυξη μιας συλλογής από εντολές, οι οποίες είναι δομημένες ώστε να εκτελούν μια συγκεκριμένη υπολογιστική εργασία. Αυτή η συλλογή λέγεται πρόγραμμα. Τα προγράμματα γράφονται από ανθρώπους σε μια γλώσσα κατανοητή σε αυτούς (π.χ. Python), μεταφράζονται σε γλώσσα μηχανής (η οποία δεν είναι κατανοητή από τον άνθρωπο) και εκτελούνται από τον υπολογιστή.

Ο προγραμματισμός συνδέεται άμεσα με την *επίλυση προβλημάτων*. Συνήθως ένα πρόγραμμα γράφεται προκειμένου να δώσει οδηγίες στον υπολογιστή για το πώς θα λύσει ένα πρόβλημα. Άρα η δουλειά του προγραμματιστή δεν είναι μόνο να γράφει κώδικα, αλλά και να σχεδιάζει λύσεις σε προβλήματα οι οποίες μπορούν να εκφραστούν ως προγράμματα.

Τα προγράμματα που εκτελούνται σε υπολογιστές λέγονται και *εφαρμογές* (applications). Μια εφαρμογή χρησιμοποιείται από χρήστες του υπολογιστή προκειμένου να διεκπεραιώσουν κάποιες εργασίες. Για παράδειγμα, μια εφαρμογή κράτησης θέσης μπορεί να βοηθά τον χρήστη να κλείσει θέση σε ένα θέατρο, μια εφαρμογή ηλεκτρονικής πληρωμής τον βοηθά να πληρώσει με τη χρεωστική του κάρτα μέσω του διαδικτύου κ.λπ. Υπάρχει πληθώρα εφαρμογών που χρησιμοποιούμε καθημερινά μέσω υπολογιστών και έξυπνων κινητών προκειμένου να συνδεθούμε με υπηρεσίες, επιχειρήσεις, το Δημόσιο, τους φίλους μας, να διεκπεραιώσουμε εργασίες και να δώσουμε ή να ανακτήσουμε πληροφορίες. Τα ηλεκτρονικά παιχνίδια είναι επίσης

εφαρμογές. Όλες αυτές οι εφαρμογές αποτελούν προϊόν προγραμματισμού και έχουν γραφτεί από προγραμματιστές.

Το συγκεκριμένο μάθημα, λοιπόν, είναι το πρώτο βήμα για να γίνει κάποιος προγραμματιστής εφαρμογών ή υπολογιστικών συστημάτων. Για να πετύχει εμπορικά μια εφαρμογή, θα πρέπει να βασίζεται σε μια καλή ιδέα. Επίσης, θα πρέπει να είναι φιλική προς τον χρήστη, γρήγορη και αποδοτική. Επομένως, ο προγραμματιστής δεν αρκεί να έχει την καλή ιδέα, αλλά χρειάζεται να διαθέτει και τις απαραίτητες δεξιότητες ώστε να τη μετατρέψει σε μια επιτυχημένη εφαρμογή. Οι προγραμματιστικές δεξιότητες μπορούν να καλλιεργηθούν με αφετηρία αυτό το βιβλίο. Η έρευση μιας καλής ιδέας χρειάζεται φαντασία, καλή εκτίμηση για την αποδοχή της από τους χρήστες και εμπειρία ώστε να υπολογιστεί η δυσκολία υλοποίησής της. Μερικοί από τους σημερινούς δισεκατομμυριούχους ήταν προγραμματιστές οι οποίοι είχαν μια καλή ιδέα και κατάφεραν να την υλοποιήσουν και να την προωθήσουν.

Το βιβλίο αυτό προσφέρει τις βάσεις για να μάθει κάποιος προγραμματισμό, αλλά κυρίως έχει στόχο να κεντρίσει το ενδιαφέρον του αναγνώστη για το συγκεκριμένο αντικείμενο. Είναι αλήθεια ότι κανείς δεν έμαθε να προγραμματίζει απλώς διαβάζοντας ένα βιβλίο ή παρακολουθώντας ένα μάθημα. Ο προγραμματισμός μαθαίνεται με πολλή εξάσκηση, πολλά σφάλματα και πολλή υπομονή. Ο προγραμματιστής επιβραβεύεται από την ευχαρίστηση να βλέπει το πρόγραμμά του να τρέχει και να λύνει το πρόβλημα για το οποίο έχει γραφτεί. Την ικανοποίηση αυτή δεν μπορεί να τη λάβει κάποιος ο οποίος έχει γράψει ένα πρόγραμμα σε χαρτί ή έχει διαβάσει ένα βιβλίο προγραμματισμού (ή έχει παρακολουθήσει ένα μάθημα). Άρα είναι απαραίτητο να περάσετε χρόνο στον υπολογιστή προγραμματίζοντας και, κυρίως, να αντλείτε ευχαρίστηση από αυτό, *σε σημείο ώστε να θέλετε να προγραμματίζετε χωρίς να σας το επιβάλλει κάποιος*.

1.1 Υπολογιστές και προγραμματισμός

Τα προγράμματα «τρέχουν» (δηλαδή εκτελούνται) σε υπολογιστές. Όμως υπολογιστές δεν θεωρούνται μόνο οι προσωπικοί υπολογιστές που βλέπουμε πάνω σε γραφεία (desktop και laptop), αλλά και τα σύγχρονα κινητά τηλέφωνα, τα tablet, οι ψηφιακές φωτογραφικές μηχανές, οι κονσόλες ηλεκτρονικών παιχνιδιών και γενικότερα οποιαδήποτε συσκευή περιέχει επεξεργαστή και μνήμη. Οι περισσότερες από αυτές τις συσκευές μπορούν να προγραμματιστούν και να εκτελέσουν εφαρμογές.

Το *υλικό* είναι τα φυσικά («χειροπιαστά») μέρη που συνθέτουν έναν υπολογιστή. Το κυριότερο μέρος είναι η κεντρική μονάδα επεξεργασίας (ΚΜΕ·

Central Processing Unit, CPU). Πρόκειται για το «μυαλό» του υπολογιστή, το οποίο μπορεί να εκτελεί απλές εργασίες όπως πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση), μετακίνηση δεδομένων από μια θέση μνήμης σε άλλη κ.ά. Η ταχύτητα ενός επεξεργαστή υπολογίζεται με τον *ρυθμό ρολογιού* του επεξεργαστή, ο οποίος μετρά πόσους κύκλους κάνει το δευτερόλεπτο. Σε έναν κύκλο μπορεί να κάνει μια βασική πράξη. Για παράδειγμα, ένας επεξεργαστής 1GHz κάνει ένα δισεκατομμύριο πράξεις το δευτερόλεπτο.

Η κύρια μνήμη (RAM) είναι μια μακρά ακολουθία από κελιά μνήμης, καθένα από τα οποία έχει μια διεύθυνση (έναν αριθμό) που προσδιορίζει τη θέση του. Κάθε κελί μνήμης έχει οκτώ δυαδικά ψηφία (1 byte). Η μνήμη είναι *προσωρινή*, καθώς τα δεδομένα χάνονται (σβήνονται) με το κλείσιμο του υπολογιστή. Η δευτερεύουσα μνήμη αποτελεί ένα μέσο στο οποίο τα δεδομένα μπορούν να αποθηκευτούν μόνιμα: η πληροφορία δεν χάνεται με το κλείσιμο του υπολογιστή. Η δευτερεύουσα μνήμη αποθηκεύει τα δεδομένα σε μορφή *αρχείων*. Παραδείγματα συσκευών δευτερεύουσας μνήμης είναι ο σκληρός δίσκος, η μνήμη flash και οι δίσκοι SSD.

Τέλος, το υλικό ενός υπολογιστή περιλαμβάνει τις περιφερειακές συσκευές οι οποίες χρησιμοποιούνται για την είσοδο δεδομένων στο υπολογιστικό σύστημα (π.χ. πληκτρολόγιο, ποντίκι) ή την παρουσίαση δεδομένων από τον υπολογιστή στον χρήστη (π.χ. οθόνη, εκτυπωτής).

Τα προγράμματα γράφονται σε αρχεία και αποθηκεύονται στη δευτερεύουσα μνήμη του υπολογιστή. Όταν εκτελείται ένα πρόγραμμα, οι εντολές του και τα όποια δεδομένα μεταφέρονται στην κύρια μνήμη. Κατόπιν, ο επεξεργαστής διαβάζει και εκτελεί τις εντολές. Επίσης, μπορεί να διαβάζει δεδομένα από τη μνήμη και να γράφει σε αυτήν κατά τη διάρκεια της εκτέλεσης του προγράμματος. Οι λειτουργίες που θα εκτελέσει ο επεξεργαστής καθορίζονται από το εκτελούμενο πρόγραμμα. Η έξοδος του προγράμματος μπορεί να γραφτεί στην κύρια μνήμη, στη δευτερεύουσα μνήμη ή σε συσκευή εξόδου.

1.2 Συγγραφή και μετάφραση ενός προγράμματος

Ο επεξεργαστής καταλαβαίνει μόνο *γλώσσα μηχανής*, δηλαδή κώδικα από απλές εντολές που ενδέχεται να διαφέρουν ανάλογα με τον επεξεργαστή. Συγκεκριμένες εντολές σε γλώσσα μηχανής εκτελούνται απευθείας στην ΚΜΕ. Δεν μπορούμε να ανοίξουμε σε κειμενογράφο ένα πρόγραμμα σε γλώσσα μηχανής και να το καταλάβουμε, γιατί βρίσκεται σε δυαδική μορφή. Άρα η σύνθεση ενός προγράμματος σε γλώσσα μηχανής είναι εξαιρετικά δύσκολη και επίπονη.

Τη λύση έρχονται να δώσουν οι *γλώσσες προγραμματισμού*, οι οποίες επιτρέπουν στον προγραμματιστή να εκφράζει το πρόγραμμα σε μια γλώσσα εύκολα κατανοητή από αυτόν. Οι γλώσσες προγραμματισμού χωρίζονται σε κατηγορίες και επίπεδα ανάλογα με την εκφραστικότητα και την αφαιρετικότητά τους. Η πιο απλή γλώσσα είναι η Assembly: ισοδυναμεί με τη γλώσσα μηχανής και απλώς αντιστοιχίζει τις εντολές της γλώσσας μηχανής (οι οποίες είναι κωδικοποιημένες σε δυαδικό κώδικα) στις αντίστοιχες εντολές που εκφράζονται με λέξεις (π.χ. LOAD, ADD) και είναι κατανοητές από τον άνθρωπο. Οι γλώσσες υψηλού επιπέδου, όπως η Python, θυμίζουν περισσότερο την ανθρώπινη γλώσσα και περιλαμβάνουν σύνθετες εντολές και κομμάτια επαναχρησιμοποιούμενου κώδικα (π.χ. συναρτήσεις, βιβλιοθήκες κ.λπ.). Το σχήμα 1.1 παρουσιάζει παραδείγματα ενός προγράμματος σε γλώσσα μηχανής, σε Assembly και σε Python.

Γλώσσα μηχανής	Γλώσσα Assembly	Γλώσσα προγραμματισμού

0111000100001111	LOAD A	# Python code
		C = A+B

1001110110110001	ADD B	
1110000100111110	STORE C	

Σχήμα 1.1: Παράδειγμα προγράμματος σε τρεις γλώσσες

Προγράμματα σε γλώσσες υψηλού επιπέδου πρέπει να *μεταφραστούν* σε κώδικα μηχανής για να εκτελεστούν από την ΚΜΕ. Ο *μεταγλωττιστής* (compiler) είναι ένα πρόγραμμα που μεταφράζει αυτόματα πηγαίο κώδικα υψηλού επιπέδου (π.χ. C++) σε γλώσσα μηχανής, ώστε ο υπολογιστής να μπορεί να τον εκτελέσει. Το αποτέλεσμα της μεταγλώττισης πολλές φορές δεν φτάνει στο βέλτιστο, αφού πρόκειται για μηχανιστική διαδικασία. Είναι όμως εντυπωσιακά καλό και πολλές φορές καλύτερο από τον κώδικα μηχανής που θα έγραφε ένας προγραμματιστής. Όπως προαναφέρθηκε, ο απευθείας προγραμματισμός σε γλώσσα μηχανής είναι εξαιρετικά δύσκολος.

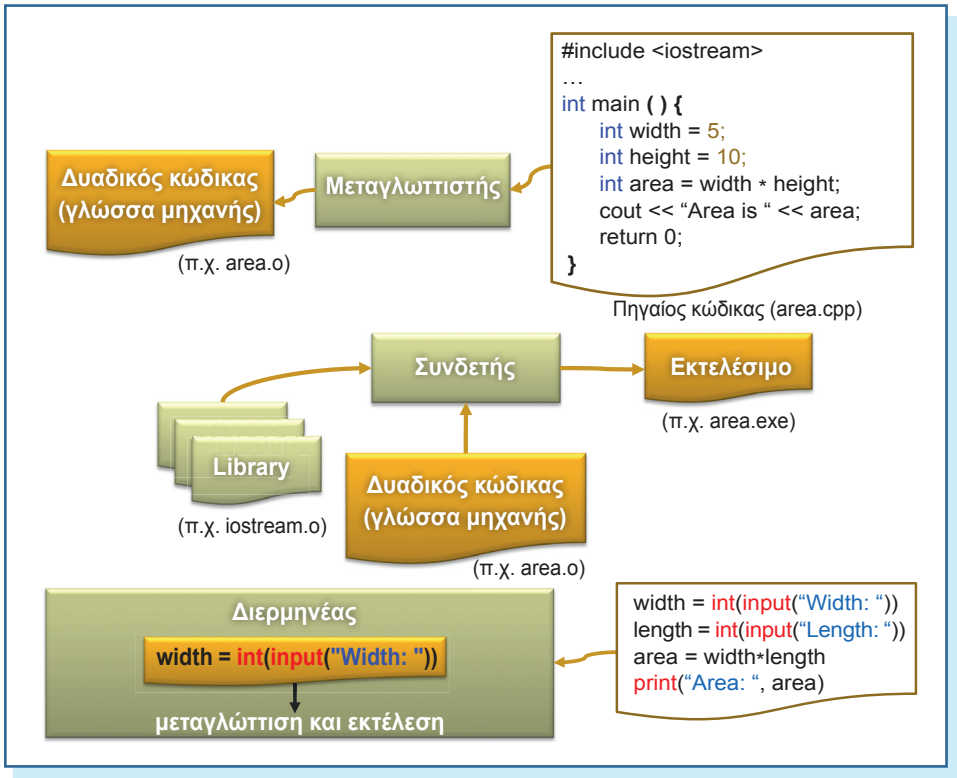
Ένα πρόγραμμα μπορεί να χρησιμοποιεί κομμάτια κώδικα (βιβλιοθήκες) που έχουν γραφτεί από άλλους (π.χ. μαθηματικές συναρτήσεις). Ο *συνδετής* (linker) είναι ένα πρόγραμμα που συνδέει ένα ή περισσότερα αρχεία (μεταγλωττισμένου) μηχανικού κώδικα σε ένα εκτελέσιμο πρόγραμμα.

Κάποιες γλώσσες προγραμματισμού (π.χ. Python) δεν έχουν μεταγλωττιστή αλλά *διερμηνέα* (interpreter). Ο διερμηνέας μεταφράζει και εκτελεί ένα πρόγραμμα εντολή προς εντολή. Δηλαδή η μετάφραση και η εκτέλεση κάθε γραμμής προγράμματος γίνεται αφότου εκτελεστεί η προηγούμενη γραμμή. Το πλεονέκτημα της διαδικασίας αυτής είναι ότι ο πηγαίος κώδικας παράγεται και ελέγχεται γρηγορότερα, αφού ο προγραμματιστής μπορεί να εκτελεί το πρόγραμμα κατευθείαν (σε ένα βήμα) χωρίς να χρειάζεται να παραχθεί πρώτα ο μεταγλωττισμένος δυαδικός κώδικας. Από την άλλη μεριά, το τελικό πρόγραμμα είναι συνήθως πιο αργό από κάποιο που έχει μεταφραστεί σε δυαδικό κώδικα. Το διάγραμμα 1.1 περιγράφει τη λειτουργία του μεταγλωττιστή, του συνδετή και του διερμηνέα.

1.3 Εκτέλεση και αποσφαλμάτωση ενός προγράμματος

Κατά τη μεταγλώττιση ή την εκτέλεση ενός προγράμματος ενδέχεται να εντοπιστούν *σφάλματα*. Το σφάλμα σε ένα πρόγραμμα λέγεται *bug* (ζωύφιο) και η διαδικασία που ακολουθείται για να διορθωθούν τα σφάλματα ονομάζεται *debugging* (αποσφαλμάτωση). Σε ένα πρόγραμμα που γράφουμε το πιθανότερο είναι να υπάρχουν σφάλματα, κυρίως λόγω απροσεξίας. Ακόμα και οι πιο έμπειροι κάνουν σφάλματα και *δεν νιώθουν ντροπή για αυτά*, αφού η ύπαρξη σφαλμάτων σε ένα μεγάλο πρόγραμμα είναι κάτι απόλυτα φυσιολογικό. Κατ' αναλογία, ακόμα και ο πιο έμπειρος συντάκτης μιας εφημερίδας μπορεί να κάνει ορθογραφικά, γραμματικά και συντακτικά λάθη. Γι' αυτό, απαιτείται *ενδελεχής έλεγχος* για την ορθότητα ενός προγράμματος. Υπάρχουν τρία είδη σφαλμάτων:

- **Συντακτικά σφάλματα** (syntax errors). Αυτά εντοπίζονται από τον μεταγλωττιστή/διερμηνέα ή τον συντακτικό αναλυτή (parser) και διορθώνονται εύκολα γιατί συνήθως ο μεταγλωττιστής συνοδεύει τον εντοπισμό τους με ακριβές μήνυμα ως προς το πρόβλημα που υπάρχει.
- **Σφάλματα κατά την εκτέλεση** (runtime errors). Αυτά συμβαίνουν κατά την εκτέλεση του προγράμματος και όχι πάντα. Ένα τυπικό σφάλμα σε αυτή την κατηγορία είναι η διαίρεση ενός αριθμού με το 0. Η εμφάνιση ενός τέτοιου σφάλματος οδηγεί το πρόγραμμα σε πρόωρο τερματισμό, καθώς κάποια δεδομένα έχουν μη έγκυρη τιμή και το πρόγραμμα δεν γνωρίζει πώς να τη χειριστεί. Ο εντοπισμός αυτών των σφαλμάτων δεν είναι πάντα εύκολος, ειδικά σε μεταγλωττισμένο (δυαδικό) κώδικα.



Διάγραμμα 1.1: Ενδεικτική λειτουργία του μεταγλωττιστή, του συνδετή και του διερμηνέα

➡ **Λογικά σφάλματα.** Όταν το πρόγραμμα παρουσιάζει λογικό σφάλμα, δεν εκτελεί την επιθυμητή λειτουργία ή παράγει λανθασμένα αποτελέσματα. Αυτά τα σφάλματα είναι δύσκολο να εντοπιστούν και να διορθωθούν – ειδικά σε μεγάλα προγράμματα, καθώς εκεί ο προγραμματιστής θα πρέπει να ελέγξει την ορθή λειτουργία όλων των τμημάτων του προγράμματος.

Να σημειωθεί ότι είναι πολύ πιο εύκολο να βρει κανείς σφάλματα σε ένα πρόγραμμα παρά να βεβαιώσει ότι είναι απόλυτα σωστό. Ένα πρόγραμμα μπορεί να εμφανίζει σφάλματα κατά την εκτέλεση μόνο για κάποιες τιμές εισόδου ή να έχει λογικά σφάλματα μόνο σε κάποιες ειδικές περιπτώσεις. Ο προγραμματιστής, λοιπόν, για να πιστοποιήσει την ορθότητα του προγράμματος, θα πρέπει να το εκτελέσει για όλες τις δυνατές εισόδους, κάτι πιθανώς αδύνατο πρακτικά. Για τον λόγο αυτό, είναι σημαντικό να ελέγχεται η ορθότητα ενός μεγάλου προγράμματος *τμηματικά*, δηλαδή να ελέγχεται *διεξοδικά* το κάθε τμήμα κώδικα που ολοκληρώνεται (π.χ. η κάθε συνάρτηση) πριν από τη συγγραφή του επόμενου τμήματος.

1.4 Κύκλος ανάπτυξης ενός προγράμματος

Ο κύκλος ανάπτυξης ενός προγράμματος περιλαμβάνει τα εξής βήματα:

1. **Σχεδιασμός.** Ο προγραμματιστής πρώτα κατανοεί τις απαιτήσεις του προγράμματος, δηλαδή τι πρέπει αυτό να επιτύχει κατά την εκτέλεσή του. Επίσης, οφείλει να προσδιορίσει την είσοδο και την επιθυμητή έξοδο του προγράμματος, δηλαδή ποια δεδομένα θα επεξεργαστεί το πρόγραμμα, με ποιον τρόπο θα τα διαβάσει, καθώς και τι θα δώσει και με ποια μορφή ως αποτέλεσμα στον χρήστη. Κατόπιν, θα πρέπει να σχεδιαστεί ο *αλγόριθμος*, δηλαδή η διαδικασία που θα ακολουθήσει το πρόγραμμα για την επίλυση του προβλήματος. Η διαδικασία αυτή θα μετατραπεί σε πρόγραμμα αφού αποφασιστούν και σχεδιαστούν οι δομές δεδομένων και οι τυχόν συναρτήσεις που θα συμπεριλαμβάνει.
2. **Συγγραφή.** Ο προγραμματιστής γράφει το πρόγραμμα στον κειμενογράφο και το αποθηκεύει ως αρχείο κειμένου με την κατάλληλη επέκταση (π.χ. «.py» για προγράμματα Python).
3. **Μεταγλώττιση και εκτέλεση.** Ο προγραμματιστής μεταγλωττίζει το πρόγραμμα σε εκτελέσιμη δυαδική μορφή και ύστερα το εκτελεί, ή το εκτελεί απευθείας με χρήση διερμηνέα.
4. **Έλεγχος και αποσφαλμάτωση.** Ο μεταγλωττιστής/διερμηνέας ενδέχεται να εντοπίσει συντακτικά σφάλματα. Επίσης, κατά την εκτέλεση μπορεί να εντοπιστούν σφάλματα τα οποία τερματίζουν αντικανονικά το πρόγραμμα (runtime errors) ή λογικά σφάλματα τα οποία πρέπει να διορθωθούν.
5. **Παράδοση.** Αφού ο προγραμματιστής διαπιστώσει ότι δεν υπάρχουν άλλα σφάλματα, παραδίδει το πρόγραμμα στον χρήστη. Εκείνος ενδεχομένως θα παρατηρήσει τυχόν δυσλειτουργίες, οι οποίες πιθανόν να οδηγήσουν σε διόρθωση (ακόμα και σε επανασχεδιασμό) του προγράμματος.

Έπειτα από κάθε βήμα στην παραπάνω διαδικασία μπορούμε να συνεχίσουμε στο επόμενο ή να μεταβούμε σε κάποιο προηγούμενο. Για παράδειγμα, αν εντοπιστούν σφάλματα στο βήμα 4, ο προγραμματιστής πρέπει να επιστρέψει στο βήμα 2 ώστε να τα διορθώσει και να επανελέγξει το πρόγραμμα.

Το *ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού* (Integrated Development Environment, IDE) είναι μια εφαρμογή που ενοποιεί τα βήματα ανάπτυξης και ελέγχου ενός προγράμματος (επεξεργασία κειμένου, μεταγλώττιση, αποσφαλμάτωση). Δημοφιλές IDE είναι το Eclipse. Η εγκατάσταση της Python συμπεριλαμβάνει το IDLE.

1.5 Τεκμηρίωση και περαιτέρω χρήση

΄Ενα πρόγραμμα γραμμένο σε γλώσσα προγραμματισμού υψηλού επιπέδου, πέραν της χρήσης του για τον σκοπό που γράφτηκε (π.χ. μια εφαρμογή), μπορεί να φανεί χρήσιμο σε άλλους προγραμματιστές. Για παράδειγμα, κάποια τμήματα του προγράμματος ενδέχεται να εκτελούν λειτουργίες οι οποίες θα μπορούσαν να χρησιμοποιηθούν από άλλα προγράμματα. Επίσης, θα μπορούσε κάποιος να πάρει το πρόγραμμα και να το αλλάξει προσαρμόζοντάς το σε μια ειδικότερη ή παραπλήσια εφαρμογή. Λόγου χάρη, ένα πρόγραμμα που γράφτηκε για να διαχειρίζεται τα δεδομένα μιας συγκεκριμένης τράπεζας υπάρχει η δυνατότητα να τροποποιηθεί και να χρησιμοποιηθεί από άλλη τράπεζα η οποία έχει παρόμοια δομή (αλλά όχι την ίδια). ΄Ενα κομμάτι κώδικα που πραγματοποιεί μια συγκεκριμένη κίνηση ενός χαρακτήρα σε κάποιο ηλεκτρονικό παιχνίδι θα μπορούσε να αλλαχθεί για να επιτύχει την κίνηση ενός άλλου χαρακτήρα (ενδεχομένως σε άλλο παιχνίδι). Τέλος, ένα πρόγραμμα πρέπει να σχεδιάζεται με τρόπο που να καθιστά δυνατή τη συντήρησή του και πιθανώς την αναδιαμόρφωσή του στο μέλλον από τον ίδιο ή από άλλους προγραμματιστές.

Είναι λοιπόν καθήκον ενός προγραμματιστή να γράφει προγράμματα όχι μόνο σωστά, αλλά και *ευανάγνωστα* και *εύληπτα*, ώστε δυνητικά να χρησιμοποιηθούν (κατά τμήματα ή ολόκληρα) και από άλλους προγραμματιστές. *Τεκμηρίωση* είναι η διαδικασία μέσω της οποίας ο προγραμματιστής διευκολύνει τον εαυτό του και άλλους προγραμματιστές να κατανοήσουν τον κώδικά του. Η τεκμηρίωση περιλαμβάνει τον καθαρισμό του κώδικα (από περιττά τμήματα που δεν χρησιμοποιούνται), την εισαγωγή σχολίων (δηλαδή επεξηγηματικού κειμένου μέσα στον κώδικα, το οποίο δεν εκτελείται) και τη συγγραφή αναφορών οι οποίες περιγράφουν τα τμήματα του κώδικα, τις λειτουργίες τους και το πώς συνδέονται μεταξύ τους. Κάποιες γλώσσες προγραμματισμού (όπως η Java) είναι ιδιαίτερα φιλικές στην τεκμηρίωση.

1.6 Καλά και κακά προγράμματα

΄Ενα καλό πρόγραμμα θα πρέπει λοιπόν να είναι *οπωσδήποτε σωστό*, αλλά και καλογραμμένο και τεκμηριωμένο, ώστε να επαναχρησιμοποιηθεί, να συντηρηθεί και να τροποποιηθεί εύκολα. Είναι μόνο αυτά τα στοιχεία ενός καλού προγράμματος; Όχι. Ειδικότερα σε γλώσσες προστακτικού προγραμματισμού (imperative programming), όπως οι C, C++, Java, Python, όπου ο προγραμματιστής καθοδηγεί τον υπολογιστή μέσω εντολών για τον τρόπο υπολογισμού του

αποτελέσματος, το πώς γράφεται ένα πρόγραμμα επηρεάζει και την ταχύτητά του. Εδώ πρέπει να γίνει κατανοητό ότι μπορούν να γραφτούν διαφορετικά σωστά προγράμματα για να λύσουν ένα πρόβλημα, όπως ακριβώς ένα αυτοκίνητο μπορεί να ακολουθήσει διαφορετικές διαδρομές για να φτάσει στον ίδιο προορισμό. Ο προγραμματιστής επιφορτίζεται με την ευθύνη να γράφει αφενός σωστά και αφετέρου *γρήγορα* προγράμματα με βάση τις δυνατότητες που του δίνει η γλώσσα προγραμματισμού. Τα τρία στοιχεία ενός καλού προγράμματος είναι λοιπόν τα εξής: (1) ορθότητα, (2) ταχύτητα, (3) καλή τεκμηρίωση. Στο μάθημα αυτό θα δώσουμε έμφαση πρωτίστως στην ορθότητα και δευτερευόντως στα άλλα δύο στοιχεία, τα οποία όμως στην πράξη είναι επίσης σημαντικά και πρέπει να λαμβάνονται υπόψη.

Πότε όμως λέμε ότι ένα πρόγραμμα είναι σωστό; Είναι σωστό αν για κάθε δυνατή είσοδο παράγει *μόνο σωστά και πλήρη αποτελέσματα*. Για παράδειγμα, αν παράγει τα αποτελέσματα μόνο στο 80% των περιπτώσεων, τότε *δεν είναι σωστό*.

1.7 Γιατί Python;

Ως πρώτη γλώσσα προγραμματισμού που μαθαίνουμε στο πρόγραμμα σπουδών επιλέξαμε την Python. Οι βασικοί λόγοι είναι οι εξής:

- Η Python είναι μια εύκολη στην εκμάθηση γλώσσα, ενώ ταυτόχρονα παρέχει πλήρη υποστήριξη στον προγραμματιστή όσον αφορά την εγκατάσταση, την εύρεση βιβλιοθηκών και βοήθειας κ.λπ. Επίσης, περιέχει όλους τους τύπους και τις δομές δεδομένων (αριθμούς, αλφαριθμητικά, λίστες, λεξικά κ.λπ.) που πιθανόν να χρειαστεί ένας προγραμματιστής για να διαχειριστεί τα δεδομένα του. Μέσω της Python μπορεί εύκολα να μάθει κανείς έλεγχο ροής προγράμματος (εντολές `if`, `for`, `while`), δομημένο προγραμματισμό (χρήση συναρτήσεων), ακόμα και αντικειμενοστρεφή προγραμματισμό (κλάσεις, αντικείμενα κ.λπ.)
- Η Python είναι εξαιρετικά δημοφιλής γλώσσα. Με βάση τον δείκτη TIOBE[1], το 2023 ερχόταν πρώτη στις προτιμήσεις των προγραμματιστών (οριακά πιο δημοφιλής από τη C) έχοντας σταθερά ανοδική πορεία τα τελευταία χρόνια. Με βάση τον ίδιο ιστότοπο, ήταν η γλώσσα της χρονιάς τα έτη 2021, 2020, 2018, 2010 και 2007.

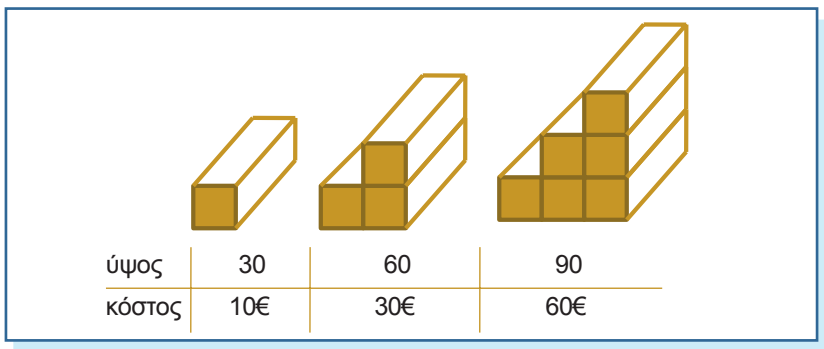
1. https://www.tiobe.com/tiobe-index/.

- Με την Python μπορεί κανείς να γράψει προγράμματα και να αναπτύξει εφαρμογές πολύ εύκολα και γρήγορα. Η Java απαιτεί χρήση αντικειμενοστρεφούς προγραμματισμού, ο οποίος δεν χρειάζεται πάντα. Έτσι, για την επίλυση του ίδιου προβλήματος, στην Python χρειάζεται πολύ λιγότερος κώδικας. Η Java και η C προϋποθέτουν τη δήλωση των μεταβλητών πριν από τη χρήση τους, ενώ στην Python αυτό δεν είναι απαραίτητο. Η C υποχρεώνει τον προγραμματιστή να δεσμεύει (και να αποδεσμεύει) μνήμη για τις σύνθετες δομές (π.χ. πίνακες), κάτι που γίνεται αυτόματα στην Python. Τέλος, η Python προσφέρει μεγάλο όγκο ελεύθερου λογισμικού (προγράμματα και βιβλιοθήκες) για πληθώρα εφαρμογών (επιστημονικούς υπολογισμούς, μηχανική μάθηση, γραφικά κ.λπ.), ώστε τελικά ο κώδικας που χρειάζεται να γράψει ο προγραμματιστής είναι πολύ λίγος. Κατά κάποιον τρόπο, η Python «κακομαθαίνει» τον προγραμματιστή, αφού του δίνει σε πολλές περιπτώσεις έτοιμες λύσεις.
- Η Python χρησιμοποιείται πλέον ευρέως και στην ανάπτυξη εμπορικών εφαρμογών.

1.8 Ένα παράδειγμα

Κλείνοντας την εισαγωγή, θα παρουσιάσουμε ένα παράδειγμα σχεδιασμού και ανάπτυξης προγράμματος σε Python. Οι προδιαγραφές (απαιτήσεις) του προγράμματος έχουν ως εξής:

> Μια κατασκευαστική εταιρεία παρέχει μια υπηρεσία δόμησης σκάλας με λίθινες ράβδους στους πελάτες της, όπως δείχνει το σχήμα 1.2. Υποθέστε ότι κάθε ράβδος έχει ύψος 30 cm και το κόστος της είναι 10€. Ζητείται ένα πρόγραμμα που θα υπολογίσει το κόστος κατασκευής μιας σκάλας δοθέντος του ύψους της.



Σχήμα 1.2: Παράδειγμα κόστους σκάλας

Κατ' αρχάς, θα πρέπει να αναλύσουμε τις απαιτήσεις του προγράμματός μας, οι οποίες είναι οι εξής:

1. Ο χρήστης θα ορίσει το επιθυμητό ύψος x της σκάλας (*είσοδος προγράμματος*).
2. Το κόστος κάθε ράβδου είναι 10€ (*δεδομένο*).
3. Το ύψος κάθε ράβδου είναι 30 cm (*δεδομένο*).
4. Ζητούμενο (*έξοδος προγράμματος*) είναι το κόστος μιας σκάλας x εκατοστών.

Κατόπιν, θα πρέπει να σχεδιάσουμε τη διαδικασία (αλγόριθμο) για την επίλυση του προβλήματος.

Μια τέτοια διαδικασία είναι η παρακάτω:

1. Διαιρώ το x με 30 για να πάρω το ύψος h της σκάλας σε ράβδους.
2. Χρειάζομαι $n = 1 + 2 + \ldots + h$ ράβδους.
3. Άρα το κόστος είναι $n \times 10$€

Τώρα γράφουμε το πρόγραμμα το οποίο λύνει το πρόβλημα:

```
x = int(input("Required height (in cm): "))
import math
h = math.ceil(x/30)
n = 0
for i in range(1,h+1):
    n = n+i
cost = n*10
print("The cost is: ", cost)
```

Το πρόγραμμα αυτό μπορεί να κατανοηθεί εύκολα από γνώστες της γλώσσας Python, αλλά επειδή δεν την έχουμε μάθει ακόμα, μην ανησυχείτε αν δυσκολεύεστε! Σε αυτή τη φάση θα προσπαθήσουμε να περιγράψουμε τη ροή του προγράμματος – οι λεπτομέρειες στις εντολές δεν έχουν σημασία. Αρχικά, στην πρώτη γραμμή το πρόγραμμα ζητά από τον χρήστη να δώσει το ύψος της σκάλας. Ύστερα, το πρόγραμμα υπολογίζει το ύψος h της σκάλας σε ράβδους (γραμμή 3). Στη συνέχεια, πραγματοποιεί μια επανάληψη η οποία υπολογίζει τον αριθμό n των απαιτούμενων ράβδων προσθέτοντας τους αριθμούς

1, 2, 3, . . . , h. Τέλος (γραμμή 7), υπολογίζει το κόστος, το οποίο τυπώνεται στην οθόνη (γραμμή 8).

Το παραπάνω πρόγραμμα γίνεται πιο γρήγορο αν παρατηρήσουμε ότι για τον υπολογισμό του αθροίσματος $n = 1 + 2 + \ldots + h$ δεν χρειάζεται να προσθέσουμε επαναληπτικά τους αριθμούς 1, 2, 3, . . . , h κάνοντας έτσι h επαναλήψεις, αλλά μπορούμε να εκμεταλλευτούμε τις γνώσεις μας στα μαθηματικά: $\sum_{i=h}^{h} i = h \times (h + 1)/2$. Αυτό σημαίνει ότι το πρόγραμμα μπορεί να γραφτεί ως εξής:

```
x = int(input("Required height (in cm): "))
import math
h = math.ceil(x/30)
n = int((h*(h+1))/2)
cost = n*10
print("The cost is: ", cost)
```

Έτσι το πρόγραμμα είναι πιο γρήγορο, γιατί αντικαθιστά μια σειρά από προσθέσεις με έναν πολλαπλασιασμό ακολουθούμενο από μια διαίρεση. Παρατηρήστε ότι το κόστος τώρα είναι ανεξάρτητο της τιμής του h, ενώ πριν ήταν ανάλογο του h. Για παράδειγμα, αν $h = 1.000$, θα χρειαζόμασταν χίλιες προσθέσεις, ενώ τώρα χρειαζόμαστε μόνο δύο πράξεις. Το παράδειγμα αυτό μας δείχνει ότι είναι σημαντικό ο προγραμματιστής να αξιοποιεί τις γνώσεις του στα μαθηματικά ώστε να επιταχύνει το πρόγραμμά του όποτε αυτό είναι δυνατό.

Ασκήσεις

1.1 Χαρακτηρίστε με Σωστό (Σ) ή Λάθος (Λ) τους παρακάτω ισχυρισμούς.

α. Μια συσκευή που δεν έχει επεξεργαστή δεν μπορεί να χαρακτηριστεί υπολογιστής.

β. Η εκτέλεση ενός προγράμματος γίνεται από την κύρια μνήμη.

γ. Ό,τι αποθηκεύεται στην κύρια μνήμη χάνεται μετά το κλείσιμο του υπολογιστή.

δ. Ένα πρόγραμμα μπορεί να αποθηκευτεί μόνο σε αρχείο κειμένου.

ε. Οι εντολές σε γλώσσα μηχανής αντιστοιχίζονται μία προς μία σε εντολές γλώσσας Assembly.

στ. Ο διερμηνέας παράγει κώδικα σε γλώσσα μηχανής.

ζ. Αν ένα πρόγραμμα δεν δίνει στην έξοδο το επιθυμητό αποτέλεσμα, τότε έχει οπωσδήποτε συντακτικά λάθη.

η. Είναι καλή πρακτική πρώτα να ολοκληρώνουμε τη συγγραφή ενός μεγάλου προγράμματος και ύστερα να το μεταγλωττίζουμε και να το εκτελούμε.

θ. Αν το προγραμμά μου τρέχει χωρίς να έχει συντακτικά λάθη ή λάθη κατά την εκτέλεση, τότε είναι σωστό.

ι. Ακόμα και ύστερα από επιτυχημένο έλεγχο και αποσφαλμάτωση, το πρόγραμμα μπορεί να χρειαστεί αλλαγές.

ια. Ο κύριος λόγος για τον οποίο τεκμηριώνουμε ένα πρόγραμμα είναι για να μπορεί να διαβαστεί και να τροποποιηθεί στο μέλλον.

ιβ. Ένα πρόγραμμα πρέπει να είναι οπωσδήποτε πολύ γρήγορο.

1.2 Δίνεται ένα πρόβλημα για το οποίο καλούνται δύο προγραμματιστές να γράψουν ένα πρόγραμμα. Το πρόγραμμα του Χρήστου δίνει πάντα σωστά αποτελέσματα. Το πρόγραμμα του Γιώργου συνήθως τερματίζει με επιτυχία και δίνει σωστά αποτελέσματα, αλλά υπάρχουν φορές που για κάποια δεδομένα εισόδου κολλάει και δεν τερματίζει ποτέ. Επίσης, το πρόγραμμα του Γιώργου είναι δύο φορές πιο γρήγορο από αυτό του Χρήστου. Ποιο από τα δύο προγράμματα είναι καλύτερο;