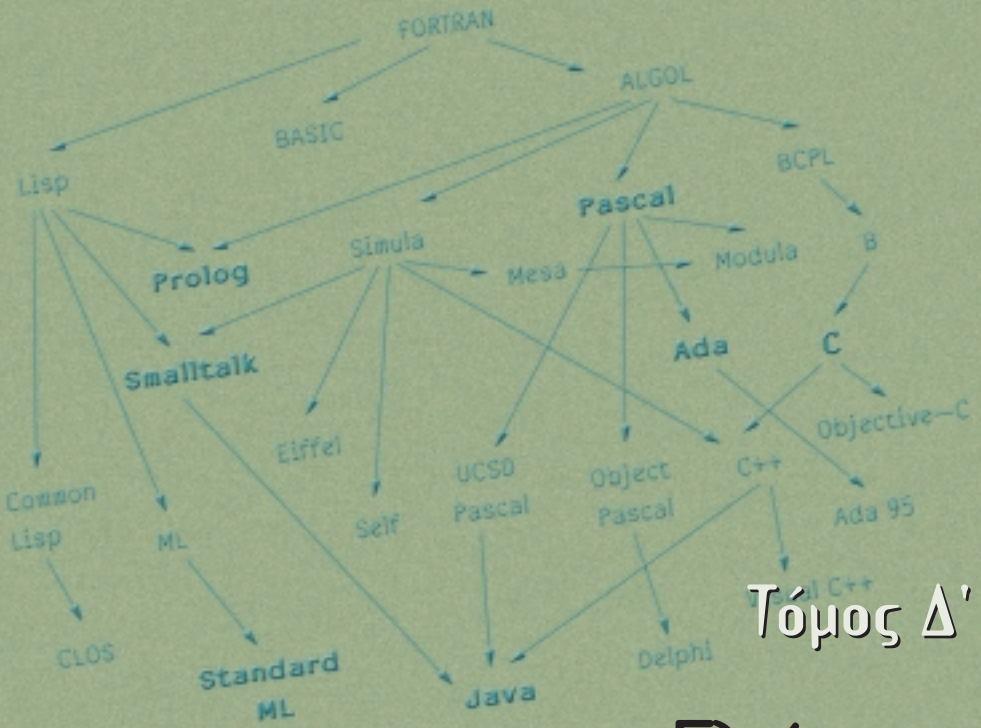
Ελληνικο Ανοικτο Πανεπιστημιο

Εισαγωγή στην Πληροφορική



Τόμος Δ'

Γλώσσες Προγραμματισμού

Κλεάνθης Θραμπουλίδης

Γλώσσες Προγραμματισμού

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχολή Θετικών Επιστημών και Τεχνολογίας

Πρόγραμμα Σπουδών
ΠΛΗΡΟΦΟΡΙΚΗ

Θεματική Ενότητα
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Τόμος Δ'

Γλώσσες Προγραμματισμού

ΚΛΕΑΝΘΗΣ ΘΡΑΜΠΟΥΛΙΔΗΣ
Επίκουρος Καθηγητής τμήματος Ηλεκτρολόγων
Μηχανικών & Τεχνολογίας Υπολογιστών
Πανεπιστημίου Πατρών

ΠΑΤΡΑ 2000

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
Σχολή Θετικών Επιστημών και Τεχνολογίας

Πρόγραμμα Σπουδών
ΠΛΗΡΟΦΟΡΙΚΗ

Θεματική Ενότητα
ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Τόμος Δ'
Γλώσσες Προγραμματισμού

Συγγραφή
ΚΛΕΑΝΘΗΣ ΘΡΑΜΠΟΥΛΙΔΗΣ
Επίκουρος Καθηγητής τμήματος Ηλεκτρολόγων
Μηχανικών & Τεχνολογίας Υπολογιστών
Πανεπιστημίου Πατρών

Κριτική Ανάγνωση
ΕΛΠΙΔΑ ΚΕΡΑΥΝΟΥ
Καθηγήτρια Τμήματος Πληροφορικής Πανεπιστημίου Κύπρου

Ακαδημαϊκός Υπεύθυνος για την επιστημονική επιμέλεια του τόμου
ΠΑΝΑΓΙΩΤΗΣ ΠΙΝΤΕΛΑΣ
Καθηγητής Τμήματος Μαθηματικών Πανεπιστημίου Πατρών

Επιμέλεια στη μέθοδο της εκπαίδευσης από απόσταση
ΓΕΡΑΣΙΜΟΣ ΚΟΥΣΤΟΥΡΑΚΗΣ

Γλωσσική Επιμέλεια
ΙΩΑΝΝΗΣ ΘΕΟΦΙΛΑΣ

Τεχνική Επιμέλεια
ΕΣΠΙ ΕΚΔΟΤΙΚΗ Ε.Π.Ε.

Καλλιτεχνική Επιμέλεια – Σελιδοποίηση
TYPORAMA

Συντονισμός ανάπτυξης εκπαιδευτικού υλικού και γενική επιμέλεια των εκδόσεων
ΟΜΑΔΑ ΕΚΤΕΛΕΣΗΣ ΕΡΓΟΥ ΕΑΠ / 1997–2000

ISBN: 960–538–084–6
Κωδικός Έκδοσης: ΠΛΗ 10/4

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

ΚΕΦΑΛΑΙΟ 2

Συντακτικό Γλώσσας

ΚΕΦΑΛΑΙΟ 3

Μεταβλητές – Σταθερές – Τύποι Δεδομένων

ΚΕΦΑΛΑΙΟ 4

Πίνακες – Δείκτες

ΚΕΦΑΛΑΙΟ 5

Τελεστές – Εκφράσεις – Προτάσεις

ΚΕΦΑΛΑΙΟ 6

Αφαιρετικότητα στις Διεργασίες

ΚΕΦΑΛΑΙΟ 7

Προτάσεις Ελέγχου Ροής

ΚΕΦΑΛΑΙΟ 10

Η Γλώσσα Προγραμματισμού Pascal

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Σκοπός

Σκοπός του κεφαλαίου είναι να προσδιορίσει το ρόλο των γλωσσών προγραμματισμού στη διαδικασία ανάπτυξης λογισμικού και να κάνει μια σύντομη αναφορά, αφενός μεν στην εξέλιξη των γλωσσών προγραμματισμού, αφετέρου δε στις σημαντικότερες κατηγορίες στις οποίες αυτές κατατάσσονται.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *περιγράψετε με δικά σας λόγια τη διαδικασία ανάπτυξης λογισμικού αναφέροντας τις κυριότερες φάσεις της,*
- *προσδιορίσετε το ρόλο που διαδραματίζει μια γλώσσα προγραμματισμού στη διαδικασία αυτή,*
- *αιτιολογήσετε γιατί η φάση της υλοποίησης δεν είναι αρκετή για την ανάπτυξη ενός συστήματος λογισμικού,*
- *χρησιμοποιήσετε τα κατάλληλα εργαλεία για τη συγγραφή πηγαίου κώδικα και την παραγωγή του αντίστοιχου εκτελέσιμου.*

Έννοιες κλειδιά

- *κύκλος ζωής λογισμικού*
- *μοντέλο ζωής λογισμικού*
- *μεθοδολογία*
- *φάση ανάλυσης*
- *φάση σχεδιασμού*
- *φάση υλοποίησης*
- *μεταγλώττιση*
- *πηγαίος κώδικας*
- *εκτελέσιμος κώδικας*
- *μοντέλο υλοποίησης*
- *μορφή προγραμματισμού*
- *προστακτικός προγραμματισμός*

Εισαγωγικές Παρατηρήσεις

Το κεφάλαιο αυτό περιλαμβάνει τρεις ενότητες. Στην πρώτη, εισάγονται βασικές έννοιες της διαδικασίας ανάπτυξης συστημάτων λογισμικού, όπως τις ορίζει ο κλάδος της επιστήμης που υποστηρίζει τη διαδικασία αυτή, και προσδιορίζεται ο ρόλος της γλώσσας προγραμματισμού. Στη δεύτερη, περιγράφε-

ται η διαδικασία ανάπτυξης προγράμματος, με ταυτόχρονη αναφορά στα εργαλεία που την υποστηρίζουν. Το κεφάλαιο ολοκληρώνεται με μια σύντομη αναφορά στην ιστορία των γλωσσών προγραμματισμού και στις διάφορες μορφές προγραμματισμού που αυτές υποστηρίζουν.

1.1 Η γλώσσα προγραμματισμού στη διαδικασία ανάπτυξης συστημάτων λογισμικού

Σκοπός

Σκοπός της ενότητας είναι να περιγράψει εν συντομία βασικές έννοιες της διαδικασίας ανάπτυξης συστήματος λογισμικού και να κάνει σαφή το ρόλο που οι γλώσσες προγραμματισμού διαδραματίζουν στη διαδικασία αυτή.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα θα μπορείτε να:

- *αναφέρετε 3 τουλάχιστον από τις φάσεις της διαδικασίας ανάπτυξης λογισμικού,*
- *αναφέρετε ποιος είναι ο στόχος των μοντέλων του κύκλου ζωής λογισμικού,*
- *αναφέρετε ποιος είναι ο στόχος των μεθοδολογιών,*
- *περιγράψετε με μια φράση τι είναι γλώσσα προγραμματισμού,*
- *δώσετε 6 τουλάχιστον παραδείγματα διεργασιών,*
- *δώσετε τον ορισμό της υπολογιστικής διεργασίας,*
- *δώσετε 4 τουλάχιστον παρα*δείγματα διεργασιών που μπορούν να αυτοματοποιηθούν.

Εισαγωγικές Παρατηρήσεις

Καθημερινά, η μία μετά την άλλη οι ανθρώπινες δραστηριότητες αυτοματοποιούνται με τη χρήση υπολογιστών. Οι υπολογιστές αυτοί, γενικού ή ειδικού σκοπού, αυτοματοποιούν πλήρως ή κατά μέρος, όλο και περισσότερες εργασίες του ανθρώπου. Απλές εργασίες, όπως συγγραφή κειμένου, έκδοση κοινοχρήστων αλλά και πιο σύνθετες, όπως πρόβλεψη καιρού και έλεγχος τηλεπικοινωνιακών κέντρων, υποστηρίζονται πλέον από υπολογιστές με τα κατάλληλα συστήματα λογισμικού. Επιχειρήσεις, αλλά ακόμη και εθνικά αμυντικά συστήματα, βασίζουν την ύπαρξή τους σε ειδικά συστήματα λογισμικού, απαιτώντας από αυτά αυξημένη αξιοπιστία. Συστήματα λογισμικού διαδραματίζουν βασικό ρόλο στην εξέλιξη όλων σχεδόν των κλάδων της επιστήμης. Το μέγεθος και η πολυπλοκότητα των σημερινών συστημάτων μεγαλώνει μέρα με τη μέρα και αντίστοιχα και οι απαιτήσεις από τη διαδικασία ανάπτυξής τους. Η ανάπτυξη συστημάτων λογισμικού έχει γίνει πλέον μια πολύ σύνθετη διαδικασία.

Μπορούμε να παραλληλίσουμε τη διαδικασία ανάπτυξης προϊόντων λογισμικού με αυτήν της ανέγερσης κτιρίων. Για την ανέγερση ενός μικρού κτιρίου, είναι πιθανόν, να μην ακολουθηθούν αυστηρές διαδικασίες και να μην εμπλακούν μηχανικοί, με αμφίβολα όμως σε κάθε περίπτωση, αποτελέσματα. Όσο όμως το μέγεθος του κτιρίου γίνεται μεγαλύτερο, τόσο πιο επιτακτική είναι η ανάγκη εμπλοκής μηχανικών στη διαδικασία ανέγερσής του. Η δημιουργία ενός συνόλου σχεδίων και προγραμματισμού δουλειάς πριν την έναρξη της κατασκευής, είναι άκρως απαραίτητη. Δεν μπορεί να διανοηθεί κανείς, την κατασκευή των σύγχρονων σημερινών κτισμάτων, χωρίς την ύπαρξη μελέτης αρχιτεκτονικών, στατικών, θερμομόνωσης και, γιατί όχι, ακόμη και πρωτοτύπου (μακέτας). Όλα αυτά, έχουν τη δική τους βαρύτητα και θέση στη διαδικασία της ανέγερσης του κτιρίου. Αντίστοιχα, για την παραγωγή ενός σύγχρονου συστήματος λογισμικού, είναι απαραίτητη η υιοθέτηση σαφώς ορισμένων βημάτων, που θα επιτρέπουν την ταυτόχρονη ικανοποίηση της πληθώρας των απαιτήσεων, που τα σύγχρονα συστήματα έχουν. Οι απαιτήσεις αυτές, που τις περισσότερες φορές είναι αντιμαχόμενες μεταξύ τους, καθιστούν το έργο του δημιουργού λογισμικού εξαιρετικά σύνθετο. Στην προσπάθειά του αυτή, έρχεται να τον στηρίξει ο κλάδος της επιστήμης του Μηχανικού Λογισμικού, που είναι γνωστός ως Τεχνολογία Λογισμικού[1] *(Software Engineering).*

[1] Χρησιμοποιείται ο όρος Τεχνολογία Λογισμικού, γιατί έχει επικρατήσει στην ελληνική βιβλιογραφία παρότι δεν αποδίδει σωστά τον αντίστοιχο αγγλικό όρο. Ένας πιο δόκιμος όρος είναι «Μηχανίκευση ή Μηχανιστική Λογισμικού».

1.1.1 Τεχνολογία Λογισμικού

Η επιστήμη της Τεχνολογίας Λογισμικού είναι η εφαρμογή επιστημονικών αρχών για το μεθοδικό μετασχηματισμό ενός προβλήματος σε μια λειτουργούσα λύση λογισμικού και για τη μετέπειτα συντήρησή του για το διάστημα που αυτό είναι χρήσιμο. Η επιστήμη αυτή, της οποίας το αντικείμενο θα μελετήσετε σε βάθος στην θεματική ενότητα 4.1 «Τεχνολογία Λογισμικού Ι», ορίζει, αφενός μεν ένα σύνολο από μοντέλα κύκλου ζωής λογισμικού (Software life cycle models), αφετέρου δε ένα σύνολο από μεθοδολογίες.

Τα **μοντέλα ζωής λογισμικού**, μεταξύ των οποίων το μοντέλο καταρράκτη (waterfall model) [Boehm ’76] και το σπειροειδές μοντέλο (spiral model) [Boehm ’88], περιγράφουν τον κύκλο ζωής λογισμικού, τη διαδικασία δηλαδή, η οποία αρχίζει από τη σύλληψη της ιδέας και τελειώνει όταν το προϊόν δεν είναι πλέον διαθέσιμο για χρήση. Οι τρεις πρώτες φάσεις, όπως τις ορίζει η πλειοψηφία των μοντέλων είναι οι εξής:

Φάση Ανάλυσης (Analysis), η οποία περιλαμβάνει την κατανόηση του προβλήματος και καταγραφή των απαιτήσεων από το σύστημα.

Φάση Σχεδιασμού (Design), η οποία περιλαμβάνει το σχεδιασμό μιας λειτουργούσας λύσης.

Φάση Υλοποίησης (Implementation), η οποία περιλαμβάνει την ανάπτυξη του εκτελέσιμου κώδικα.

Αντίθετα, μια **μεθοδολογία** ορίζει τα επιμέρους βήματα που πρέπει να εκτελέσει ο δημιουργός λογισμικού σε μια συγκεκριμένη συνήθως φάση της διαδικασίας ανάπτυξης. Για παράδειγμα, η μεθοδολογία της Δομημένης Ανάλυσης (Structured Analysis/SA) ορίζει τον τρόπο με τον οποίο προσεγγίζουμε τη φάση της κατανόησης και καταγραφής απαιτήσεων του συστήματος.

1.1.2 Ιστορική αναδρομή στη διαδικασία ανάπτυξης λογισμικού

Η διαδικασία που οι δημιουργοί λογισμικού ακολουθούσαν τα πρώτα χρόνια της εμφάνισης των υπολογιστών, χαρακτηρίζεται πλήρως από το όνομα code–and–fix που αποδόθηκε στο μοντέλο το οποίο αργότερα την αναπαράστησε. Η διαδικασία χαρακτηρίζεται από δύο βασικά βήματα:

- συγγραφή κώδικα και
- διόρθωση των προβλημάτων του κώδικα.

Καμία ενέργεια δε γινόταν για απαιτήσεις, σχεδιασμό, έλεγχο και συντήρη-

ση, με αποτέλεσμα σύντομα να γίνουν εμφανή τα προβλήματα της διαδικασίας. Το σημαντικότερο από αυτά, ήταν ότι μετά από ένα αριθμό διορθώσεων (επεμβάσεων) στον κώδικα, αυτός έχανε κάθε μορφή δόμησης με απαγορευτικά πλέον κόστη συντήρησης. Η διαπίστωση του προβλήματος αυτού οδήγησε στην υιοθέτηση μίας φάσης πριν την κωδικοποίηση η οποία ονομάστηκε **φάση σχεδιασμού**. Στη συνέχεια, διαπιστώθηκε ότι ακόμη και καλοσχεδιασμένα συστήματα ήταν πολλές φορές άχρηστα, γιατί δεν κάλυπταν τις απαιτήσεις του χρήστη. Αυτό οδήγησε στην υιοθέτηση μιας επιπλέον φάσης πριν από την φάση του σχεδιασμού, της **φάσης της ανάλυσης**. Το αποτέλεσμα ήταν περί το 1959 να διακριθούν οι παραπάνω φάσεις, διαμορφώνοντας το *κατά στάδια μοντέλο* (stage–wise), από το οποίο προήλθε το *μοντέλο του καταρράκτη* (waterfall model). Παράλληλα, διαπιστώθηκε ότι το κόστος διόρθωσης του κώδικα ήταν πολύ μεγάλο, εξαιτίας της μηδαμινής προετοιμασίας για έλεγχο και τροποποιήσεις. Η διαπίστωση αυτή, οδήγησε σε αναγνώριση της ανάγκης για προετοιμασία για τις ενέργειες αυτές πριν από τη συγγραφή κώδικα.

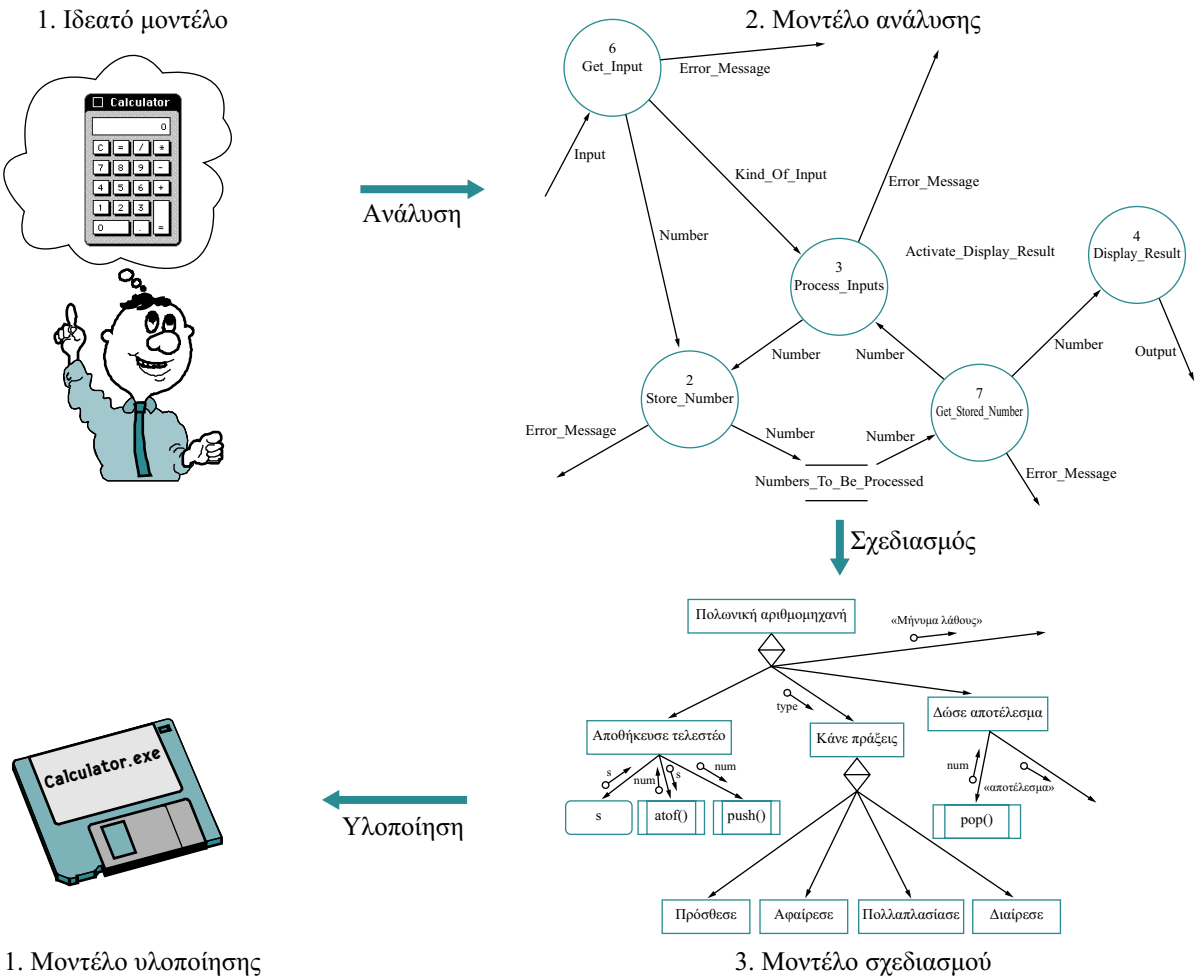
1.1.3 Η διαδικασία ανάπτυξης λογισμικού σαν διαδικασία παραγωγής μοντέλων

Τα παραπάνω οδηγούν στην αναπαράσταση της διαδικασίας ανάπτυξης λογισμικού όπως αυτή δίνεται στο σχήμα 1.1. Σύμφωνα με την αναπαράσταση αυτή, η διαδικασία ξεκινά από το ονομαζόμενο *ιδεατό μοντέλο* (conceptual model) και ολοκληρώνεται με τη δημιουργία του *μοντέλου υλοποίησης* (implementation model). Το ιδεατό μοντέλο, είναι αυτό που έχει στο μυαλό του ο χρήστης πριν αρχίσει τη διαδικασία ανάπτυξης του συστήματος, ενώ το μοντέλο υλοποίησης είναι σε μορφή που μπορεί να εκτελέσει ο υπολογιστής και αυτή δεν είναι άλλη από την *γλώσσα μηχανής* (machine language)[1]. Η μετάβαση από το ιδεατό μοντέλο στο μοντέλο υλοποίησης γίνεται σταδιακά και με την παρεμβολή δύο άλλων μοντέλων: του μοντέλου ανάλυσης και του μοντέλου σχεδιασμού.

Το μοντέλο σχεδιασμού, μεταξύ των άλλων, ορίζει επακριβώς τις επιμέρους υπολογιστικές διεργασίες που πρέπει να εκτελέσει ο υπολογιστής για την ικανοποίηση των καταγεγραμμένων απαιτήσεων του χρήστη, καθώς και τα επί μέρους βήματα της κάθε διεργασίας[2]. Μπορούμε να πούμε ότι είναι μια

[1] Μια σημειογραφία που άμεσα κατανοεί ο υπολογιστής.

[2] Η διεργασία αποτελεί το βασικό δομικό στοιχείο των παραδοσιακών μεθοδολογιών ανάπτυξης συστημάτων λογισμικού.



Σχήμα 1.1.

Μοντέλα στη διαδικασία ανάπτυξης λογισμικού.

περιγραφή του τρόπου επίλυσης ενός προβλήματος[1] ή εκτέλεσης μιας εργασίας. Για την περιγραφή των επί μέρους βημάτων της κάθε διεργασίας, η επιστήμη του Software Engineering προτείνει ένα σύνολο από τεχνικές, μεταξύ των οποίων τα *δομημένα αγγλικά* (structured English) ή τα αντίστοιχα *δομημένα ελληνικά*, που εμείς θα χρησιμοποιήσουμε στα πλαίσια αυτού του βιβλίου, τα *διαγράμματα ροής* (flow charts) καθώς και άλλες τεχνικές που θα γνωρίσετε στην θεματική ενότητα 4.1. Τέλος, η φάση της υλοποίησης έχει σαν στόχο τη μετατροπή του μοντέλου σχεδιασμού σε μοντέλο υλοποίησης. Στη μετατροπή αυτή, που θα δούμε αναλυτικά στη συνέχεια, η γλώσσα προγραμματισμού διαδραματίζει το σημαντικότερο ρόλο.

[1] Η προσέγγιση ή η μέθοδος που χρησιμοποιείται για την επίλυση ενός προβλήματος αποτελεί τον αλγόριθμο

1.1.4 Η διεργασία σαν βασικό δομικό στοιχείο στη διαδικασία ανάπτυξης λογισμικού

Στην καθημερινή μας ομιλία χρησιμοποιούμε πολύ συχνά ονόματα διεργασιών. Η παρακάτω φράση «Παρακαλώ φτιάξε ένα νες μέτριο με γάλα και σερβίρισέ το στο 5» χρησιμοποιεί τα ονόματα των διεργασιών «φτιάξε ένα νες» και «σερβίρισε» για να αναθέσει την εκτέλεση του έργου που οι διεργασίες αυτές προσδιορίζουν στο άτομο στο οποίο απευθύνεται η πρόταση.

Ο όρος «διεργασία» αναφέρεται σε ένα σύνολο βημάτων που μπορεί να εκτελέσει μια οντότητα του φυσικού κόσμου για να φέρει εις πέρας ένα απλό ή σύνθετο έργο. Στη συγκεκριμένη περίπτωση που η διεργασία εκτελείται από έναν υπολογιστή ονομάζεται **«υπολογιστική διεργασία»**. Σαν παραδείγματα διεργασιών μπορούν να αναφερθούν οι παρακάτω:

1. Η διεργασία του υπολογισμού του μέσου όρου δεδομένων αριθμών.
2. Η διεργασία της ανεύρεσης των πελατών μιας επιχείρησης που έχουν υπερβεί το όριο πίστωσης.
3. Η διεργασία της ανεύρεσης των φοιτητών της δανειστικής βιβλιοθήκης, οι οποίοι έχουν υπερβεί το χρονικό όριο δανεισμού.
4. Η διεργασία του καθαρισμού ρούχων.
5. Η διεργασία της ετοιμασίας φαγητού.

Ορισμένες από τις παραπάνω διεργασίες μπορούν να ανατεθούν στον υπολογιστή όπως οι 1, 2 και 3, άλλες όπως οι 4 και 5 όχι, τουλάχιστον μέχρι σήμερα. Η διεργασία καθαρισμού των ρούχων, για παράδειγμα, μπορεί να υποστηριχθεί εν μέρει από μια μηχανή.

Η περιγραφή μιας διεργασίας πρέπει να δίνεται σαν ένα σύνολο από βήματα σε μορφή που μπορεί να κατανοήσει και επομένως, να εκτελέσει η οντότητα στην οποία ανατίθεται η εκτέλεσή της. Για παράδειγμα, αν η διεργασία πρόκειται να εκτελεστεί από ένα άνθρωπο, αυτή θα πρέπει να περιγραφεί σε μία από τις ομιλούμενες γλώσσες (Ελληνική, Αγγλική, κ.λπ.). Αν η διεργασία πρόκειται να εκτελεστεί από έναν υπολογιστή, η περιγραφή θα πρέπει να γίνει σε μία σημειογραφία που *άμεσα* ή *έμμεσα* κατανοεί ο υπολογιστής. Άμεσα κατανοεί μόνο τη γλώσσα μηχανής, αλλά η περιγραφή μιας διεργασίας με τη γλώσσα αυτή είναι σχεδόν αδύνατη για τα σημερινά, πολύπλοκα και αυξημένων απαιτήσεων, συστήματα. Επομένως, η μόνη επιλογή είναι αυτή της έμμεσης κατανόησης. Λέμε ότι ο υπολογιστής κατανοεί έμμεσα τις γλώσσες προγραμματισμού γιατί μια περιγραφή με αυτές μπορεί αυτόματα να μεταγλωττιστεί σε γλώσσα

μηχανής και, άρα, να γίνει κατανοητή από τον υπολογιστή. Μπορούμε να πούμε λοιπόν, ότι μια *γλώσσα προγραμματισμού* είναι μία συστηματική σημειογραφία (notation) με την οποία περιγράφουμε υπολογιστικές διεργασίες.

Άσκηση Αυτοαξιολόγησης 1.1

Θεωρήστε τη διαδικασία υπολογισμού της μέσης θερμοκρασίας του μήνα από τις μέσες ημερήσιες θερμοκρασίες. Καταγράψτε βασικές υπολογιστικές διεργασίες που ένα σύστημα πρέπει να εκτελέσει για να την φέρει σε πέρας.

1.2 Η φάση της υλοποίησης

Σκοπός

Σκοπός της ενότητας είναι να περιγράψει τα επί μέρους βήματα που πρέπει να ακολουθήσετε, για να δημιουργήσετε ένα πρόγραμμα που θα μπορεί να εκτελέσει ο υπολογιστής σας.

Προσδοκώμενα Αποτελέσματα

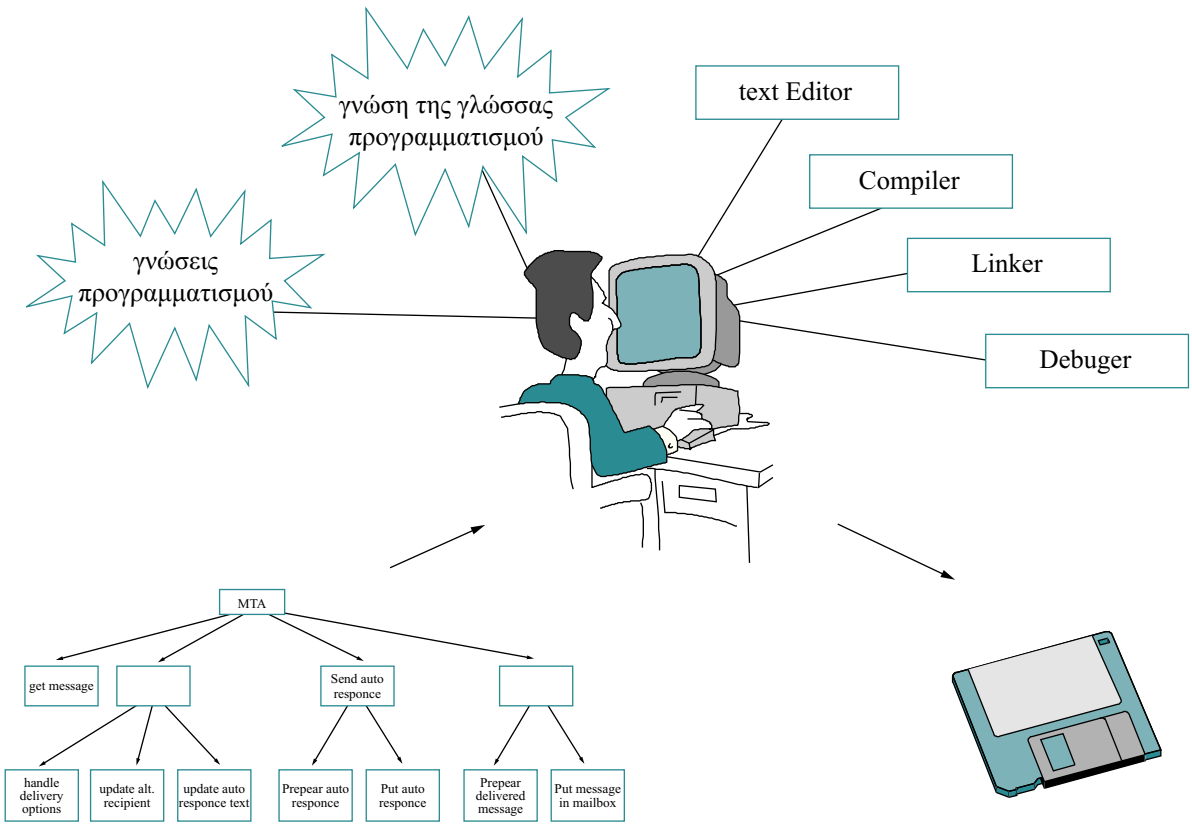
Όταν θα έχετε ολοκληρώσει την ενότητα θα μπορείτε να:

- *κατασκευάσετε τον εκτελέσιμο κώδικα για έναν πηγαίο κώδικα που θα σας δοθεί,*
- *αναφέρετε τα βασικά βήματα για τη δημιουργία του εκτελέσιμου κώδικα,*
- *αναφέρετε ποια εργαλεία χρησιμοποιούνται στη διαδικασία αυτή,*
- *εξηγήσετε το ρόλο της βιβλιοθήκης στη διαδικασία αυτή.*

Εισαγωγικές Παρατηρήσεις

Η φάση της υλοποίησης μπορεί να θεωρηθεί σαν μία σύνθετη διεργασία, η οποία λαμβάνει σαν είσοδο το μοντέλο σχεδιασμού[1] *και έχει σαν έξοδο τον εκτελέσιμο κώδικα. Η διεργασία αυτή μπορεί να ανατεθεί σε μια οντότητα του πραγματικού κόσμου για να την φέρει σε πέρας. Η οντότητα αυτή είναι συνήθως ο προγραμματιστής, ο οποίος, όπως φαίνεται στο σχήμα 1.2, χρησιμοποιεί τη γνώση της γλώσσας προγραμματισμού, τις γενικότερες γνώσεις του για προγραμματισμό και τον υπολογιστή σαν εργαλείο για να τον υποστηρίξει στο δύσκολο αυτό έργο. Στη συνέχεια, θα περιγράψουμε τα επιμέρους βήματα που πρέπει να ακολουθήσει ο προγραμματιστής, για τη δημιουργία του εκτελέσιμου κώδικα.*

[1] Για απλά προβλήματα το μοντέλο αυτό δημιουργείται στο μυαλό του προγραμματιστή.



Σχήμα 1.2

Ο προγραμματιστής εκτελώντας την διεργασία της φάσης της υλοποίησης.

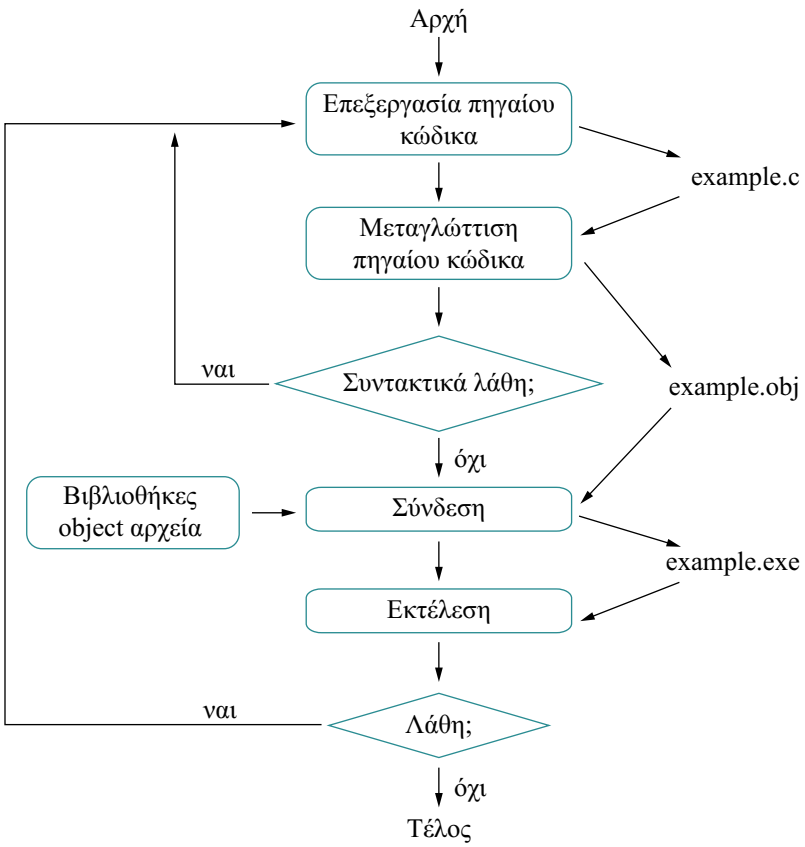
1.2.1 Τα βήματα της φάσης της υλοποίησης

Το σχήμα 1.3 αναπαριστά με την τεχνική σχεδιασμού flow chart, τη διεργασία της φάσης της υλοποίησης. Μπορούμε να διακρίνουμε τις παρακάτω βασικές επί μέρους διεργασίες:

1. συγγραφή πηγαίου κώδικα
2. μεταγλώττιση πηγαίου κώδικα
3. σύνδεση τμημάτων που αποτελούν το μοντέλο υλοποίησης (εκτελέσιμο κώδικα)
4. έλεγχος καλής λειτουργίας

Οι διεργασίες αυτές, που διαφοροποιούνται ανάλογα με το υπολογιστικό σύστημα αλλά και το συγκεκριμένο μεταγλωττιστή, είναι πλήρως ή εν μέρει αυτοματοποιημένες. Για παράδειγμα, η διεργασία της συγγραφής κώδικα υποστηρίζεται εν μέρει από τον υπολογιστή, απαιτεί όμως ουσιαστική συμ-

μετοχή του προγραμματιστή[1]. Αντίθετα, η διεργασία της μεταγλώττισης είναι πλήρως αυτοματοποιημένη και απαιτεί από τον προγραμματιστή μόνο την ενεργοποίηση του μεταγλωττιστή. Σε κάθε περίπτωση, για τη διεξαγωγή στην πράξη της φάσης της υλοποίησης, απαιτείται αναφορά στα εγχειρίδια του λειτουργικού συστήματος για οδηγίες σχετικές με τη μεταγλώττιση και σύνδεση (linking) αλλά και στα αντίστοιχα εγχειρίδια του μεταγλωττιστή για πληροφορίες σχετικά με τη διαδικασία της μεταγλώττισης.



Σχήμα 1.3

Η διεργασία της φάσης της υλοποίησης

1.2.2 Συγγραφή πηγαίου κώδικα

Στο πρώτο αυτό βήμα, ο προγραμματιστής επιλέγοντας μία από τις διαθέσιμες γλώσσες προγραμματισμού και χρησιμοποιώντας έναν επεξεργαστή κειμένου[2] (text editor), δημιουργεί ένα αρχείο που περιέχει τον *πηγαίο κώδικα* του προγράμματός του. Η C επιβάλει την αποθήκευση του πηγαίου

[1] Τα περισσότερα CASE εργαλεία υποστηρίζουν αυτόματη παραγωγή μέρους του πηγαίου κώδικα από το μοντέλο σχεδιασμού.

[2] Προσέξτε υπάρχει διαφορά μεταξύ text editor και word processor. Έχει επικρατήσει να αποδίδονται με τον ίδιο ελληνικό όρο επεξεργαστής–κειμένου.

κώδικα[1] σε ένα αρχείο με επέκταση .c. Στα πλαίσια του παραδείγματός μας, γράψτε τον παρακάτω πηγαίο κώδικα[2] χωρίς να ασχοληθείτε με τη σημασία του πηγαίου κώδικα. Αποθηκεύστε τον στο αρχείο example.c.

```
/* A Simple C Program */
#include <stdio.h>
main( )
{
printf("hello, world");I
}
```

1.2.3 Μεταγλώπιση

Η διεργασία της μεταγλώττισης είναι, όπως αναφέραμε, πλήρως αυτοματοποιημένη. Ο προγραμματιστής αρκεί να ενεργοποιήσει τον μεταγλωττιστή ενημερώνοντάς τον με το όνομα του αρχείου που θέλει να μεταγλωττίσει και με ένα σύνολο από παραμέτρους που καθορίζουν τις απαιτήσεις του από τον μεταγλωττιστή. Η ενεργοποίηση του μεταγλωττιστή γίνεται απλά εκτελώντας το πρόγραμμα του μεταγλωττιστή, όπως το λειτουργικό σας σύστημα ορίζει. Για τον τρόπο με τον οποίο μπορείτε να ενημερώσετε τον μεταγλωττιστή για τις ιδιαίτερες απαιτήσεις που έχετε από αυτόν, π.χ., παραγωγή κώδικα με βελτιστοποίηση ως προς το χρόνο εκτέλεσης ή ως προς το μέγεθος του κώδικα, ή συμπερίληψη πληροφορίας εκσφαλμάτωσης κ.λπ., ενημερωθείτε από το εγχειρίδιο χρήσης του ή τις on–line οδηγίες του. Σε ένα γραφικό περιβάλλον, τα παραπάνω εκτελούνται συνήθως μέσα από τις επιλογές ενός μενού που παρέχει το περιβάλλον ανάπτυξης. Αντίθετα, σε ένα παραδοσιακό περιβάλλον με γραμμή εντολής (command line) η ενεργοποίηση του μεταγλωττιστή έχει τη μορφή:

<όνομα μεταγλωττιστή> [<λίστα παραμέτρων>] <όνομα αρχείου>

Για παράδειγμα, για το μεταγλωττιστή της WATCOM[3] στο MSDOS, η εντολή

wcc example

---

[1] Σε επόμενο κεφάλαιο, θα δούμε ότι ο πηγαίος κώδικας οργανώνεται σε περισσότερα του ενός αρχεία, καθώς και τους λόγους που επιβάλουν μια τέτοια οργάνωση.

[2] Το κλασικό πια παράδειγμα που ο Dennis Ritchie είχε στην αντίστοιχη ενότητα του ιστορικού κειμένου [Kernighan ’74].

[3] Πρόκειται για μια εκπαιδευτική έκδοση του εμπορικού μεταγλωττιστή Watcom C/C++.

ενεργοποιεί το μεταγλωττιστή, ο οποίος μεταγλωττίζει το αρχείο example.c, και σε περίπτωση μη ανεύρεσης λαθών, τυπώνει στην οθόνη τα παρακάτω:

```
WATCOM C Optimizing Compiler Version 8.5e
Copyright by WATCOM Systems Inc. 1984, 1991. All rights reserved.
WATCOM is a trademark of WATCOM Systems Inc.
example.c: 4 lines, 0 warnings, 0 errors
Code size: 17
```

και παράγει το αρχείο example.obj.

Αντίθετα, στην περίπτωση ανεύρεσης λαθών, το αποτέλεσμα της διεργασίας της μεταγλώττισης είναι μια λίστα με τα λάθη που εντόπισε ο μεταγλωττιστής. Τα λάθη αυτά έχουν σχέση με το συντακτικό της γλώσσας και είναι γνωστά σαν συντακτικά λάθη (syntax errors). Παρακάτω, δίνεται η αναφορά του μεταγλωττιστή για την περίπτωση που λείπει η αγκύλη στο τέλος του πηγαίου κώδικα.

```
example.c(4): Error! E1077: Missing '}'
example.c: 4 lines, 0 warnings, 1 errors
```

Στην περίπτωση αυτή, θα πρέπει να εντοπίσετε τα λάθη, με τη βοήθεια της πληροφορίας που αναφέρει ο μεταγλωττιστής, και χρησιμοποιώντας τον text editor να τα διορθώσετε. Στη συνέχεια, επαναλαμβάνετε τη διαδικασία της μεταγλώττισης μέχρι την επιτυχή έκβασή της.

1.2.4 Σύνδεση

Η διεργασία της σύνδεσης που ακολουθεί είναι και αυτή πλήρως αυτοματοποιημένη και εκτελείται από τον υπολογιστή με τη βοήθεια του προγράμματος του *συνδέτη* (Linker). Ο προγραμματιστής θα πρέπει να ενεργοποιήσει το συνδέτη και να ορίσει τις παραμέτρους της διεργασίας της σύνδεσης. Στα περισσότερα περιβάλλοντα ανάπτυξης, ο συνδέτης καλείται αυτόματα μετά από επιτυχή μεταγλώττιση χωρίς τη μεσολάβηση του προγραμματιστή[1].

Το αποτέλεσμα της σύνδεσης είναι η δημιουργία του εκτελέσιμου κώδικα ή η αναφορά τυχόν προβλημάτων, όπως για παράδειγμα, αδυναμία εντοπισμού μιας συνάρτησης ή μιας εξωτερικής μεταβλητής. Ο εκτελέσιμος κώδικας αποθηκεύεται σε αρχείο που έχει το όνομα του αρχείου πηγαίου κώδικα και επέκταση .exe (για Dos και Windows).

---

[1] Ενημερωθείτε για το τρόπο λειτουργίας του δικού σας συστήματος.

Ο συνδέτης έχει τη δυνατότητα να συνδέσει περισσότερα του ενός αρχεία αντικείμενου κώδικα (object code), των οποίων μάλιστα, ο αντίστοιχος πηγαίος κώδικας δεν είναι αναγκαστικά στην ίδια γλώσσα προγραμματισμού. Αναζητεί επιπλέον, σε βιβλιοθήκες τα σώματα των συναρτήσεων που ο προγραμματιστής χρησιμοποίησε στο πρόγραμμά του (Στο θέμα αυτό θα αναφερθούμε αναλυτικά στο κεφάλαιο περί συναρτήσεων. Δες ένθετο πλαίσιο για βιβλιοθήκη).

Για τον WATCOM μεταγλωττιστή, η εντολή

```
wcl example
```

έχει σαν αποτέλεσμα την ενεργοποίηση του μεταγλωττιστή και στη συνέχεια του συνδέτη, ο οποίος δίνει μια αναφορά της παρακάτω μορφής:

```
WATCOM Linker Version 7.0
Copyright by WATCOM Systems Inc. 1985, 1991. All rights reserved.
WATCOM is a trademark of WATCOM Systems Inc.
loading object files
searching libraries
creating a DOS executable
```

και παράγει το αρχείο example.exe, το οποίο αποτελεί το μοντέλο υλοποίησης, ή αλλιώς, τον εκτελέσιμο κώδικα του προγράμματος, τον οποίο μπορείτε να εκτελέσετε για να διαπιστώσετε την ορθή λειτουργία του. Η πρόταση

```
example
```

στη γραμμή διαταγών του DOS δίνει εντολή στο λειτουργικό να φορτώσει και εκτελέσει το αρχείο example.exe. Στην οθόνη θα έχετε το παρακάτω αποτέλεσμα:

```
hello, world
```

Τα πιθανά λάθη που θα εμφανιστούν, εντοπίζονται, διορθώνονται και η διεργασία μεταγλώττισης και σύνδεσης επαναλαμβάνεται μέχρι την επιτυχή λειτουργία του προγράμματος.

Μεταγλώττιση

Μεταγλώττιση είναι η διαδικασία της μετάφρασης υψηλού επιπέδου περιγραφής, όπως περιγραφή σε Basic, Pascal, C, σε γλώσσα μηχανής που αποτελεί μορφή κατανοητή από τον υπολογιστή. Η μεταγλώττιση, αποτελεί μια σύνθετη και χρονοβόρο για τον άνθρωπο διεργασία και για το λόγο αυτό την ανέθεσε στον υπολογιστή.

Βιβλιοθήκη

Κατά τη συγγραφή προγραμμάτων ο προγραμματιστής έχει τη δυνατότητα να επαναχρησιμοποιεί (reuse) έτοιμα κομμάτια κώδικα, τα οποία είτε έχει ο ίδιος αναπτύξει και χρησιμοποιήσει στο παρελθόν είτε προσφέρονται από άλλους δημιουργούς λογισμικού. Τα έτοιμα αυτά τμήματα κώδικα είναι συνήθως με τη μορφή μονάδων (συναρτήσεων, κλάσεων) με την κάθε μονάδα να εκτελεί μια συγκεκριμένη διεργασία, η οποία όσο γενικότερη είναι τόσο μεγαλύτερες είναι οι πιθανότητες επαναχρησιμοποίησής της. Οι μονάδες αυτές είναι οργανωμένες κατά λογικές κατηγορίες με κάθε λογική κατηγορία να συνθέτει μια βιβλιοθήκη που χαρακτηρίζεται από ένα όνομα. Κάθε μεταγλωττιστής της C συνοδεύεται από τη βασική βιβλιοθήκη της γλώσσας (standard C library), η οποία περιέχει ένα σύνολο από συναρτήσεις που υλοποιούν πολύ βασικές διεργασίες. Μια τέτοια συνάρτηση είναι η printf, η οποία υλοποιεί τη διεργασία της εξόδου μορφοποιημένης πληροφορίας στην κύρια έξοδο του υπολογιστή (συνήθως οθόνη). Στη C, για να χρησιμοποιήσετε μια συνάρτηση της βασικής βιβλιοθήκης είναι απαραίτητο να συμπεριλάβετε στον πηγαίο κώδικα ένα ή περισσότερα *αρχεία επικεφαλίδας* (header files), όπως ορίζει το εγχειρίδιο χρήσης της βιβλιοθήκης για κάθε συνάρτησή της. Για παράδειγμα, για τη χρήση της printf πρέπει να περιλάβετε στην αρχή του πηγαίου κώδικα την πρόταση #include <stdio.h>.

1.2.5 Διαδικασία ανάπτυξης παραδειγμάτων

Για την ανάπτυξη των προγραμμάτων που χρησιμοποιούνται σαν παραδείγματα στα επόμενα κεφάλαια του βιβλίου, θα ξεκινάμε αμέσως τη συγγραφή του πηγαίου κώδικα χωρίς να ασχολούμαστε καθόλου, στις περισσότερες περιπτώσεις, με τα μοντέλα ανάλυσης και σχεδιασμού. Και αυτό γιατί, αφενός μεν στόχος της θεματικής αυτής ενότητας είναι ο προγραμματισμός, αφετέρου δε δεν έχετε ακόμη τις απαραίτητες γνώσεις για να δημιουργήσετε τα μοντέλα ανάλυσης και σχεδιασμού, αντικείμενο που θα καλύψετε σε άλλη θεματική ενότητα, αργότερα. Επιπλέον, τα παραδείγματά μας είναι συνήθως απλά και δεν απαιτούν από τη φύση τους, την επισταμένη δημιουργία των μοντέλων αυτών. Σε ορισμένες περιπτώσεις, πιθανόν να χρησιμοποιήσουμε τα δομημένα ελληνικά σαν πρώτη μορφή περιγραφής πριν από τη δημιουργία πηγαίου κώδικα. Ας σημειωθεί πως αυτό το τελευταίο αποτελεί μέρος της φάσης σχεδιασμού.

Άσκηση Αυτοαξιολόγησης 1.2

Τοποθετήστε τις παρακάτω διεργασίες που έχουν σχέση με την ανάπτυξη και εκτέλεση ενός προγράμματος σε σωστή χρονική σειρά.

1. αποθήκευση πηγαίου κώδικα
2. σύνδεση
3. αναφορά λαθών από μεταγλωττιστή
4. εκτέλεση προγράμματος
5. συγγραφή πηγαίου κώδικα
6. μεταγλώττιση
7. εμφάνιση αποτελεσμάτων προγράμματος
8. αναφορά λαθών από συνδέτη

Δραστηριότητα 1.1

Ακολουθήστε τη διαδικασία δημιουργίας εκτελέσιμου κώδικα για το πρόγραμμα που έχει σαν πηγαίο κώδικα τον παρακάτω.

```
#include <stdio.h>
main()
{
int num1 = 10;
int num2 = 20;
int sum;
sum = num1 + num2;
printf("το άθροισμα του %d με το %d είναι: %d", num1, num2, sum);
}
```

Στη συνέχεια, εκτελέστε το για να ελέγξετε την ορθή λειτουργία του. Το αποτέλεσμα της εκτέλεσης μπορείτε να δείτε στο τέλος του βιβλίου.

Δραστηριότητα 1.2

Αναζητήστε στον υπολογιστή σας το ευρετήριο όπου έχει εγκατασταθεί ο C μεταγλωττιστής σας. Εντοπίστε και καταγράψτε τα εκτελέσιμα αρχεία του μεταγλωττιστή και του συνδέτη. Κάντε το ίδιο για τις βιβλιοθήκες που τον συνοδεύουν. Τέλος, αναζητήστε τα αρχεία επικεφαλίδας και δείτε το περιεχόμενο του αρχείου math.h. Σχολιάστε το. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

1.3 Μορφές προγραμματισμού και ιστορία γλωσσών

Σκοπός

Ο σκοπός της ενότητας είναι να κάνει μια σύντομη αναφορά στις εναλλακτικές μορφές προγραμματισμού (programming paradigms)[1] και, ταυτόχρονα, μια ιστορική αναδρομή στην γλώσσα προστακτικού προγραμματισμού C, αιτιολογώντας παράλληλα την επιλογή της στα πλαίσια αυτού του βιβλίου.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα, θα μπορείτε να :

- *αναφέρετε 4 τουλάχιστον μορφές προγραμματισμού,*
- *περιγράψετε τη διαφορά μεταξύ προστακτικού και δηλωτικού προγραμματισμού,*
- *αναφέρετε τουλάχιστον 4 γλώσσες που υποστηρίζουν την προστακτική μορφή προγραμματισμού,*
- *αναφέρετε τη βασική διαφορά μεταξύ των γλωσσών Pascal και Prolog,*
- *αναφέρετε 6 τουλάχιστον λόγους για τους οποίους επιλέχθηκε να χρησιμοποιηθεί η C στο παρόν βιβλίο.*

1.3.1 Μορφές προγραμματισμού

Μορφή ή στυλ προγραμματισμού (programming paradigm/programming style) είναι μια συλλογή από έννοιες, οι οποίες προσδιορίζοντας έναν ορισμένο τρόπο σκέψης και, άρα, έκφρασης της λύσης, επηρεάζουν το σχεδιασμό των προγραμμάτων. Εάν μπορούμε να δομήσουμε τη λύση ενός προβλήματος με τις βασικές έννοιες μιας μορφής προγραμματισμού, τότε μόνο μπορούμε να χρησιμοποιήσουμε μια γλώσσα προγραμματισμού που υποστηρίζει τη συγκεκριμένη μορφή προγραμματισμού για να υλοποιήσουμε τη λύση. Ορισμένες γλώσσες (βλέπε Πίνακα 1.1), εισήγαγαν νέα στυλ προγραμματισμού, νέους δηλαδή τρόπους σκέψης στον προγραμματισμό.

Οι μορφές που υποστηρίζουν προγραμματισμό υψηλού επιπέδου, μπορούν να διακριθούν σε τρεις κατηγορίες [Ambler 92]:

[1] Στην ελληνική βιβλιογραφία χρησιμοποιείται ευρέως ο όρος μοντέλα προγραμματισμού.

Πίνακας 1.1

Γλώσσες που εισήγαγαν βασικά στυλ προγραμματισμού.

Γλώσσα	Στυλ προγραμματισμού
FORTRAN	προστακτικό
Lisp	συναρτησιακό
Simula	αντικειμενοστρεφές
Prolog	λογικό

- **Operational,** με βασικό χαρακτηριστικό την βήμα προς βήμα περιγραφή της πορείας εξεύρεσης μιας λύσης. Στην κατηγορία αυτή ανήκουν ο *προστακτικός* (imperative), ο *αντικειμενοστρεφής* (object–oriented) καθώς και ο *συναρτησιακός* (functional) προγραμματισμός.
- **Demonstrational**, γνωστή και σαν programming by example[1], είναι η μορφή προγραμματισμού που επεξηγεί τη λύση με χρήση συγκεκριμένων παραδειγμάτων και αφήνει στο σύστημα να γενικεύσει για άλλες περιπτώσεις. Ορισμένοι τη θεωρούν υποκατηγορία της operational μορφής προγραμματισμού. Σαν παραδείγματα γλωσσών αυτής της κατηγορίας αναφέρονται η PT [Hsia 88] και η Metamouse [Maulsby 89].
- **Definitional**, με βασικό χαρακτηριστικό την έκθεση των χαρακτηριστικών της λύσης, τα οποία συνήθως περιορίζουν το πεδίο τιμών της, χωρίς αυτό να συνοδεύεται από περιγραφή του τρόπου εξεύρεσής της. Στην κατηγορία αυτή, ανήκουν ο λογικός προγραμματισμός (logic programming), ο βασισμένος σε περιορισμούς προγραμματισμός (constraint programming), ο λογικός βασισμένος σε περιορισμούς προγραμματισμός (Constraint Logic Programming) και μια όψη του συναρτησιακού προγραμματισμού[2].

Ο συναρτησιακός προγραμματισμός περιλαμβάνει την έννοια της *συνάρτησης* (function) σαν βασικό δομικό στοιχείο. Στην πράξη, οι γλώσσες που υποστηρίζουν αυτό το στυλ προγραμματισμού περιλαμβάνουν και δομικά στοιχεία προστακτικού προγραμματισμού. Επίσης, οι περισσότερες προστακτικές και αντικειμενοστρεφείς γλώσσες έχουν υιοθετήσει κάποια μορφή συνάρτησης, χωρίς όμως αυτό να σημαίνει πως υποστηρίζουν πλήρως τη συναρ-

[1] Programming by Example Home Page – http://lcs.www.media.mit.edu/people/lieber/PBE/.

[2] Ένα καλό παράδειγμα σύγχρονης συναρτησιακής γλώσσας αποτελεί η Haskell.

τησιακή μορφή προγραμματισμού. Στην πράξη, μία γλώσσα προγραμματισμού σπάνια υποστηρίζει μια μόνο μορφή προγραμματισμού. Συνήθως, δανείζεται έννοιες από περισσότερες μορφές και σαν αποτέλεσμα, υποστηρίζει περισσότερα από ένα στυλ προγραμματισμού. Για παράδειγμα, η C++ υλοποιεί βασικές έννοιες από το προστακτικό και το αντικειμενοστρεφές παράδειγμα. Η ML από το συναρτησιακό και το προστακτικό, η CLOS αποτελεί επέκταση της συναρτησιακής γλώσσας Lisp εισάγοντας έννοιες της αντικειμενοστρεφούς μορφής, η Prolog++ αποτελεί επέκταση της γλώσσας λογικού προγραμματισμού Prolog εισάγοντας έννοιες του αντικειμενοστρεφούς παραδείγματος, κ.ο.κ.

Γλώσσες όπως οι Pascal, Ada, Algol, Fortran και C υποστηρίζουν την προστακτική μορφή προγραμματισμού. Ο προγραμματιστής με τις γλώσσες αυτές προσδιορίζει, βήμα προς βήμα την πορεία επίλυσης του προβλήματος. Χρησιμοποιεί μεταβλητές, για να αναπαραστήσει τα δεδομένα και ενέργειες (actions), που αποτελούν τις βασικές μονάδες του προστακτικού στυλ προγραμματισμού, για να αλλάξει τις τιμές των μεταβλητών [Sethi ’97]. Αντίθετα, με την Prolog ο προγραμματιστής δουλεύει στην definitional ή declarative κατηγορία μορφών προγραμματισμού. Σύμφωνα με αυτή, ο προγραμματιστής δεν περιγράφει στον υπολογιστή πώς να λύσει το πρόβλημα, αλλά τον εφοδιάζει με όλη τη διαθέσιμη πληροφορία που είναι απαραίτητη σε κάποιον για να λύσει το πρόβλημα. Το σύστημα χρησιμοποιεί αυτή την πληροφορία με το δικό του τρόπο για να οδηγηθεί στην ανεύρεση της λύσης[1]. Η προμήθεια ενός χάρτη της περιοχής για την ανεύρεση της πορείας για συγκεκριμένο προορισμό, αποτελεί χαρακτηριστικό παράδειγμα δηλωτικής (declarative) μορφής, σε αντίθεση με τις σαφείς οδηγίες κατεύθυνσης (π.χ., τρία τετράγωνα παρακάτω δεξιά, μετά από δύο τετράγωνα αριστερά, κ.λπ.) που αποτελούν προστακτική μορφή.

Μία μορφή προγραμματισμού που, τα τελευταία χρόνια, γνωρίζει μεγάλη εξάπλωση γιατί υπόσχεται να δώσει (και ήδη δίνει) λύσεις σε πολλά από τα προβλήματα της διαδικασίας ανάπτυξης λογισμικού και όχι μόνο[2], είναι η *Αντικειμενοστρεφής Προσέγγιση*[3] (ΑΠ). Η ΑΠ θεωρεί το σύστημα σαν συνά-

[1] Αυτό γίνεται για παράδειγμα διαμέσου του ενσωματωμένου inference Engine της Prolog.

[2] Βλέπε αντικειμενοστρεφή συν–σχεδιασμό (co–design) υλικού–λογισμικού.

[3] Στην πράξη υπάρχει αντικειμενοστρεφής προστακτική αλλά και αντικειμενοστρεφής δηλωτική μορφή.

θροιση *αντικειμένων* (objects) διαφόρων *κατηγοριών* (classes),τα οποία αλληλεπιδρούν μεταξύ τους για την παροχή των εξυπηρετήσεών του. Ο αντικειμενοστρεφής προστακτικός προγραμματισμός, πρώτα ορίζει τα κατάλληλα για το πρόβλημα αντικείμενα και, στη συνέχεια, τα χρησιμοποιεί για να περιγράψει βήμα προς βήμα την εξεύρεση της λύσης. Παραδείγματα γλωσσών που υποστηρίζουν τη μορφή αυτή προγραμματισμού, που αποτελεί το αντικείμενο της θεματικής ενότητας 6.4, είναι οι Smalltalk, C++, Object Pascal και Java.

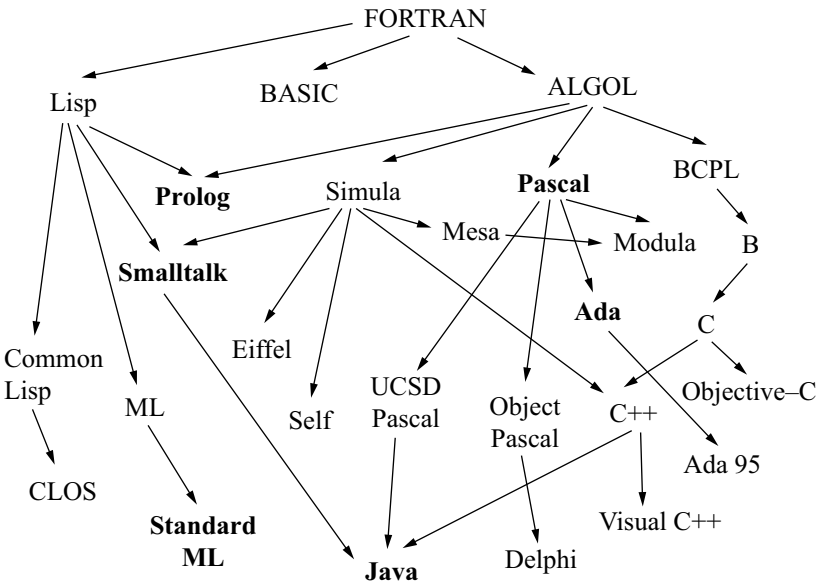
1.3.2 Ιστορία των γλωσσών προστακτικού προγραμματισμού

Η ιστορία των γλωσσών προστακτικού προγραμματισμού υψηλού επιπέδου, αρχίζει το 1957 με την εμφάνιση της FORTRAN (FORmula TRANslation), μιας γλώσσας φτιαγμένης για μαθηματικούς υπολογισμούς. Η γλώσσα εξακολουθεί και σήμερα να αποτελεί την πιο δημοφιλή επιλογή μεταξύ των επιστημόνων και των μηχανικών. Η διάδοχος της FORTRAN, η ALGOL, κυριάρχησε στη δεκαετία του ’60 σε βαθμό που η κατηγορία των προστακτικών γλωσσών να αναφέρεται σαν ALGOL οικογένεια, από όπου και ο όρος ALGOL–like. Στην πράξη, η γλώσσα περισσότερο θαυμάστηκε παρά υιοθετήθηκε, αντίθετα με τις απογόνους της, Pascal και C, που υιοθετήθηκαν ευρέως. Αμφότερες εμφανίστηκαν στις αρχές της δεκαετίας του ’70 και, η μεν Pascal σχεδιάστηκε σαν εκπαιδευτική γλώσσα από τον Nicklaus Wirth, η δε C σαν γλώσσα προγραμματισμού συστήματος (system programming) από τον Dennis Ritchie. Και οι δύο, εξελίχθηκαν σε γενικού σκοπού (general purpose) προστακτικές γλώσσες και επηρέασαν σε μεγάλο βαθμό επόμενες γλώσσες προστακτικού προγραμματισμού μεταξύ των οποίων οι Modula, Concurrent Pascal και Ada (από την Pascal) και C++ και Java (από την C).

Μεταξύ των γλωσσών που υιοθετήθηκαν σε μεγάλο βαθμό θα πρέπει να αναφέρουμε και την COBOL (Common Business Oriented Language) με μεγάλη απήχηση στον επιχειρηματικό κόσμο, καθώς και την BASIC (Beginners All–purpose Symbolic Instruction Code) που, αν και σχεδιάστηκε για εκπαίδευση αρχαρίων στο προγραμματισμό, εξελίχθηκε σε γενικού σκοπού γλώσσα. Το σχήμα 1.4, παρουσιάζει ένα τμήμα του γενεαλογικού δένδρου των γλωσσών προγραμματισμού, όπου μπορείτε να δείτε τις σημαντικότερες γλώσσες προγραμματισμού και τον τρόπο με τον οποίο αυτές επηρέασαν την εξέλιξη των υπολοίπων.

Για μια σύντομη αναδρομή στην ιστορία των γλωσσών προγραμματισμού

σάς συνιστώ να ανατρέξετε στο [Horowitz '84] ή [Horowitz '95] ή στο [Sethi '97]. Για ακόμη περισσότερα, μπορείτε να αναφερθείτε στο [Wexelblat 81] ή στο πιο πρόσφατο [Bergin '96]. Επίσης, στη διεύθυνση http://cuiwww.unige.ch/cgi–bin/langlist και, κάτω από τον τίτλο «The Language List», μπορείτε να δείτε μία προσπάθεια δημιουργίας μίας λίστας όλων των γνωστών γλωσσών προγραμματισμού.



Σχήμα 1.4.

Μέρος του γενεαλογικού δένδρου των γλωσσών προγραμματισμού.

Δραστηριότητα 1.3

Ανατρέξτε στη βιβλιογραφία τη σχετική με την ιστορία των γλωσσών προγραμματισμού και γράψτε ένα κείμενο (1 με 2 σελίδες), στο οποίο θα αναφέρεστε ιστορικά στις σημαντικότερες, κατά τη γνώμη σας, γλώσσες προγραμματισμού και στον τρόπο με τον οποίο αυτές επηρέασαν την εξέλιξη των υπολοίπων.

1.3.3 Ιστορία της C

Η γλώσσα C σχεδιάστηκε και αναπτύχθηκε το 1972 στα AT&T Bell Labs από τον Dennis Ritchie. Βασίστηκε στην BCPL [M. Richard, 1967] αλλά και στην απόγονο της B [Thompson '72], την οποία ανέπτυξε ο Ken Thompson και το διάστημα εκείνο χρησιμοποιούσε στην ανάπτυξη της νέας έκδοσης του λειτουργικού συστήματος UNIX[1]. Ο Dennis Ritchie, προσθέτοντας και αφαιρώντας στοιχεία από την B με στόχο να την απομακρύνει από το υλικό,

[1] Οι μέχρι τότε εκδόσεις του UNIX ήταν γραμμένες σε γλώσσα Assembly.

δημιούργησε μια νέα γλώσσα που την ονόμασε C (η γλώσσα μετά την B).

Η C όντας ευέλικτη και αποδοτική, χρησιμοποιήθηκε το 1973 για να ξαναγραφεί το μεγαλύτερο τμήμα του UNIX σε έναν PDP–11. Στη συνέχεια, και για χρόνια, χρησιμοποιήθηκε αποκλειστικά για system programming στο UNIX. Η πρώτη επίσημη τεκμηρίωση της γλώσσας έκανε την εμφάνισή της μόλις το 1977, με τίτλο «The C Programming Language» από τους Brian Kernighan και Dennis Ritchie. Η τεκμηρίωση αυτή αποτέλεσε για χρόνια το «ευαγγέλιο» των προγραμματιστών της C και είναι γνωστό σαν «white book» ή K&R πρότυπο [Kernighan '78].

Με την πάροδο των χρόνων, η γλώσσα άρχισε να χρησιμοποιείται και σε άλλα πεδία εφαρμογών εκτός του system programming για το οποίο σχεδιάστηκε, κατακτώντας ένα πολύ μεγάλο μέρος της αγοράς, με αποτέλεσμα να θεωρείται στις αρχές τις δεκαετίας του '90 μία από τις επικρατέστερες γλώσσες. Σε αυτό, συνέτειναν και τα πολλά πλεονεκτήματά της, όσον αφορά ευελιξία, αποδοτικότητα, φορητότητα και ταχύτητα εκτέλεσης, σε σύγκριση με άλλες ανάλογες γλώσσες όπως Fortran, Basic, Pascal. Βέβαια, η γλώσσα γνώρισε πολλές αλλαγές από την πρώτη και γνωστή σαν K&R έκδοση. Οι αλλαγές αυτές οδήγησαν στη δεύτερη έκδοση της γλώσσας που είναι γνωστή σαν ANSI C πρότυπο [ANSI '88] [Kernighan '88]. Μια πολύ καλή αναφορά στην εξέλιξη της γλώσσας γίνεται στο άρθρο [Ritchie '93].

1.3.4 Γιατί C;

Η επιλογή της C για τη διδασκαλία της προστακτικής μορφής προγραμματισμού στο παρόν βιβλίο, έγινε για ένα σύνολο από λόγους που παρουσιάζονται παρακάτω:

- Είναι σχετικά μικρή και εύκολη στην εκμάθηση.
- Υποστηρίζει top–down και modular σχεδιασμό αλλά και δομημένο (structured) προγραμματισμό.
- Είναι αποτελεσματική (efficient) παράγοντας συμπαγή και γρήγορα στην εκτέλεση προγράμματα.
- Είναι φορητή (portable), ευέλικτη (flexible) και ισχυρή (powerful).
- Δε βάζει περιορισμούς, γεγονός πάντως που πολλές φορές αποβαίνει σε βάρος της.
- Αποτελεί με την C++ την ευρύτερα χρησιμοποιούμενη γλώσσα σε ερευνητικά και αναπτυξιακά προγράμματα.

- Υπάρχει μια πολύ μεγάλη εγκατεστημένη βάση εφαρμογών που αναπτύχθηκαν με τη γλώσσα αυτή και πρέπει να συντηρούνται και να εξελίσσονται.
- Η C μπορεί να χρησιμοποιηθεί σαν χαμηλού επιπέδου γλώσσα προγραμματισμού επιτρέποντας άμεση πρόσβαση στους πόρους του υπολογιστή και άρα στην αποτελεσματική και χωρίς overhead αξιοποίησή τους. Ταυτόχρονα, μπορεί να χρησιμοποιηθεί και σαν γλώσσα υψηλού επιπέδου, καθώς η πληθώρα των διαθέσιμων βιβλιοθηκών υπερκαλύπτει τις απαιτήσεις ανάπτυξης λογισμικού εφαρμογής (Application Software).
- Η γνώση της C αποτελεί ένα πολύ καλό εφόδιο για την εκμάθηση της Java.

Σύνοψη

Στο κεφάλαιο αυτό, αναφέραμε τις βασικές έννοιες της διαδικασίας ανάπτυξης λογισμικού και προσδιορίσαμε το ρόλο που διαδραματίζει η γλώσσα προγραμματισμού στη διαδικασία αυτή. Παρά το γεγονός ότι, τα πρώτα χρόνια της ζωής των ηλεκτρονικών υπολογιστών η γλώσσα προγραμματισμού αποτέλεσε το κύριο εργαλείο για την ανάπτυξη λογισμικού, στη συνέχεια, αυτή περιορίστηκε σαν βασικό εργαλείο μόνο της φάσης της υλοποίησης, επηρεάζοντας βέβαια σημαντικά και τη φάση του σχεδιασμού.

Η ιστορία των γλωσσών προγραμματισμού έχει να παρουσιάσει μια μεγάλη ποικιλία γλωσσών. Ορισμένες από αυτές, όπως οι Fortran, Lisp, Simula και Prolog, εισήγαγαν νέα στυλ προγραμματισμού, νέους δηλαδή τρόπους σκέψης στη φάση του προγραμματισμού. Άλλες πάλι αποτελούν απογόνους επιτυχημένων γλωσσών όπως η C++ που αποτελεί απόγονο της C στην οποία πρόσθεσε τα πλεονεκτήματα της Simula.

Η οικογένεια των γλωσσών του προστακτικού προγραμματισμού ξεκινά με τη Fortran. C και Pascal αποτελούν επιτυχημένες γενικού σκοπού γλώσσες αυτής της οικογένειας και επιλέχτηκαν να χρησιμοποιηθούν —στα πλαίσια του βιβλίου— για την παρουσίαση των βασικών στοιχείων του προστακτικού προγραμματισμού. Ιδιαίτερα, χρησιμοποιείται η C, η οποία, αφενός μεν κυριάρχησε την περασμένη δεκαετία, αφ' ετέρου δε αποτέλεσε τη βάση για γλώσσες που κυριαρχούν (C++) ή διαφαίνεται ότι θα κυριαρχήσουν (Java) την επόμενη δεκαετία.

Βιβλιογραφία κεφαλαίου

[ANSI '88]

Περισσότερες πληροφορίες για το πρωτότυπο ISO/IEC 9899 μπορείτε να βρείτε στο δια–δίκτυο στη διεύθυνση http://wwwold.dkuug.dk/JTC1/SC22/WG14/.

[Bergin '96]

Bergin Thomas, Gibson Richard, *«History of Programming Languages»*, Addison Wesley, 1996.

Το βιβλίο αποτελεί σημαντική προσπάθεια συγκέντρωσης πολλών σημαντικών γλωσσών και συνιστάται για κάθε αναγνώστη που εμπλέκεται στη χρήση ή ανάπτυξη γλωσσών προγραμματισμού. Είναι βασισμένο στα πρακτικά του δεύτερου ACM SIGPLAN συνεδρίου *«History of Programming Languages»*. Περιλαμβάνει περίληψη των πρακτικών του συνεδρίου, καθώς και άρθρα των σημαντικότερων συντελεστών στο χώρο των γλωσσών προγραμματισμού: Frederick Brooks, Alain Colmerauer, Richard Gabriel, Ralph Griswold, Per Brinch, Hansen, Alan Kay, C. H. Lindsey, Barbara Liskov, Richard Nance, Elizabeth Rather, Dennis Ritchie, Jean Sammet, Guy Steele, Bjarne Stroustrup, William Whitaker, και Niklaus Wirth. Μεταξύ των γλωσσών που ιδιαίτερα αναφέρονται είναι οι C, C++, Smalltalk,Pascal, Ada, Prolog, Lisp, ALGOL 68, FORMAC, CLU, Icon, Forth και Concurrent Pascal.

[Boehm '76]

Boehm W. Barry, *«Software Engineering»* IEEE Trans. Comput., vol. C–25, pp 1226–1241, Dec. 1976.

[Boehm '88]

Boehm W. Barry, *«A Spiral Model of Software Development and Enhancement»* IEEE Computer, vol. 21, no 5, p.61–72, May 1988.

[Horowitz '84]

Horowitz Ellis, *«Βασικές αρχές γλωσσών προγραμματισμού»*, Εκδόσεις Κλειδάριθμος, 1993.

Το βιβλίο είναι μετάφραση της δεύτερης Αμερικάνικης έκδοσης του *«Fundamentals of Programming Languages»* Horowitz Ellis, Computer Science Press, 1984

[Horowitz ’95]

Horowitz Ellis, *«Fundamentals of Programming Languages»*, Third edition, Computer Science Press, 1995.

[Hsia ’88]

Hsia, Y. T., Ambler A., *«Programming Through Pictorial transformations»,* Proc. Int’l Conf. Computer Languages ’88, IEEE CS Press, Los Alamitos, Calif., Order No. 1988, pp. 10–16.

[Kernighan ’88]

Η δεύτερη έκδοση του βιβλίου *«The C Programming Language»* κυκλοφόρησε το 1988 καλύπτοντας πλέον την ANSI C.

Αποτελεί το πλέον έγκυρο βιβλίο μετά το ISO/IEC 9899, πολύ καλό για αναφορά αλλά πολύ δύσκολο για να χρησιμοποιηθεί για εκμάθηση της γλώσσας. Ελληνική έκδοση σε μετάφραση από τον Κλειδάριθμο 1990.

[Maulsby ’89]

Maulsby D. L., Witten H., *«Inducing Programs in a Direct–Manipulation Environment»,* Proc. CHI’89, ACM Press, New York, 1989, pp. 57–62.

[Ritchie ’93]

Ritchie M. Dennis, *«The Development of the C Language»*, ACM SIGPLAN Notices, March ’93, p. 207.

Το άρθρο δίνει μια πολύ καλή αναφορά στην εξέλιξη της γλώσσας C.

[Sethi ’97]

Sethi Ravi, *«Programming Languages: Concepts and Constructs»* 2nd Edition, Addison Wesley, 1996. Reprinted with corrections, April, 1997.

[Wexelblat ’81]

Wexelblat L. Richard, *«History of Programming Languages»,* Los Angeles, 1981.

Περιέχει τα πρακτικά του πρώτου ACM SIGPLAN συνεδρίου «History of Programming Languages» Los Angeles on June 1–3, 1978. Καταγράφει τις γλώσσες προγραμματισμού που δημιουργήθηκαν το τέλος της δεκαετίας ’60 (1967) και παρέμεναν σε χρήση μέχρι το 1977, επη-

ρεάζοντας σημαντικά την εξέλιξη του προγραμματισμού. Περιέχονται άρθρα ερευνητών που διαδραμάτισαν σημαντικό ρόλο στην ανάπτυξη και χρήση των γλωσσών: ALGOL, APL, APT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/I, SIMULA, AND SNOBOL.

ΚΕΦΑΛΑΙΟ 2

Συντακτικό Γλώσσας

Σκοπός

Ο σκοπός του κεφαλαίου είναι να εισάγει τις βασικές έννοιες του συντακτικού των γλωσσών προγραμματισμού, καθώς και τους βασικούς περιορισμούς που αυτό θέτει στη διαδικασία συγγραφής προγραμμάτων. Για την καλύτερη κατανόηση των εννοιών, γίνεται αναφορά στο συντακτικό της γλώσσας C.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *αναφέρετε τουλάχιστον 2 τρόπους περιγραφής του συντακτικού μιας γλώσσας προγραμματισμού,*
- *αναφέρετε τουλάχιστον 3 σύνολα χαρακτήρων στα οποία οι γλώσσες προγραμματισμού βασίζουν το λεξιλόγιό τους,*
- *αναφέρετε τουλάχιστον 3 επί μέρους κατηγορίες των λέξεων που απαρτίζουν το λεξιλόγιο μιας γλώσσας,*
- *αναφέρετε τουλάχιστον 4 κατηγορίες δεσμευμένων λέξεων της ANSI C,*
- *διακρίνετε σε ένα C πρόγραμμα τις λέξεις–κλειδιά,*
- *διακρίνετε τα συντακτικά από τα σημασιολογικά λάθη,*
- *αναφέρετε τουλάχιστον 4 κανόνες δημιουργίας ονομάτων στη γλώσσα C,*
- *αναφέρετε τουλάχιστον 3 κανόνες για τη δημιουργία ευανάγνωστου προγράμματος.*

Έννοιες κλειδιά

- *συντακτικό γλώσσας*
- *BNF*
- *ASCII κώδικας*
- *Unicode*
- *δεσμευμένη λέξη (reserved word)*
- *λέξη–κλειδί (keyword)*
- *ευανάγνωστο πρόγραμμα*
- *συντακτικό λάθος (syntax error)*
- *σημασιολογικό λάθος (semantic error)*
- *case sensitive*

Εισαγωγικές Παρατηρήσεις

Ως χρήστες της ελληνικής γλώσσας, χρησιμοποιείτε το λεξιλόγιό της για να σχηματίζετε προτάσεις και, στη συνέχεια, παραγράφους που απαρτίζουν τα κείμενά σας. Αντίστοιχα, ένας χρήστης μιας γλώσσας προγραμματισμού, ένας προγραμματιστής όπως θα τον αποκαλούμε στη συνέχεια, χρησιμοποιεί το λεξιλόγιο της γλώσσας για να συντάξει προτάσεις που απαρτίζουν τα προγράμματά του.

Γνωρίζετε επιπλέον, ότι η δημιουργία κειμένου υπόκειται σε ένα σύνολο από κανόνες που είναι γνωστοί σαν γραμματικοί και συντακτικοί κανόνες. Θυμόσαστε σίγουρα την επιμονή της δασκάλας σας στους κανόνες αυτούς. «Δε θα μάθετε να γράφετε χωρίς γραμματική και συντακτικό» χρειαζόταν να επαναλαμβάνει πολλές φορές. Δυστυχώς, αυτό συμβαίνει και σε κάθε γλώσσα προγραμματισμού. Η γνώση του λεξιλογίου και των συντακτικών κανόνων αποτελεί βασική προϋπόθεση για τη συγγραφή προγραμμάτων. Στο κεφάλαιο αυτό, θα δώσουμε τις βασικές έννοιες του συντακτικού, ενώ στα επόμενα, θα καλύψουμε το μεγαλύτερο κομμάτι του λεξιλογίου και της σύνταξης της C.

2.1 Τι είναι το αλφάβητο μιας γλώσσας προγραμματισμού;

Όπως κάθε ομιλούμενη γλώσσα (π.χ., ελληνική, αγγλική, κ.λπ.) χρησιμοποιεί τους χαρακτήρες του αλφαβήτου της για να σχηματίσει λέξεις, έτσι και κάθε γλώσσα προγραμματισμού έχει το δικό της αλφάβητο. Με τους χαρακτήρες του αλφαβήτου της, η γλώσσα σχηματίζει τις λέξεις ή τα σύμβολα που αποτελούν το λεξιλόγιο της γλώσσας.

Το αλφάβητο της γλώσσας C, σύμφωνα με το ANSI πρότυπο [ANSI '88], αποτελείται από 96 χαρακτήρες. Οι χαρακτήρες αυτοί είναι ο χαρακτήρας του κενού, οι χαρακτήρες ελέγχου οριζόντιου στηλοθέτη (tab), κάθετου στηλοθέτη, αλλαγής σελίδας (form feed) και νέας γραμμής (new–line), και οι παρακάτω 91 γραφικοί χαρακτήρες:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
_ { } [ ] # ( ) < > % : ; . ? * + – / ^ & | ~ ! = , \ " '
```

Οι χαρακτήρες αυτοί, χρησιμοποιούνται από τη C για τη δημιουργία των λέξεων και των ειδικών συμβόλων που αποτελούν το λεξιλόγιό της. Αποτελούν δε, μέρος ενός ευρύτερου συνόλου χαρακτήρων που είναι γνωστό σαν ASCII (American Standard Code for Information Interchange) κώδικας[1]. Ένα εναλλακτικό σύνολο χαρακτήρων που χρησιμοποιείται κύρια από την IBM στους μεγάλους υπολογιστές της, είναι το EBCDIC (Extended Binary Coded Decimal Interchange Code), ενώ ένα τρίτο, το πλέον σύγχρονο, έχει το όνομα unicode και λύνει πολλά από τα προβλήματα που δημιούργησαν τα προηγούμενα [Unicode].

2.2 Η έννοια του συντακτικού

Ένα σύνολο κανόνων ορίζουν τον τρόπο με τον οποίο οι λέξεις μιας γλώσσας προγραμματισμού μπορούν να συγκροτήσουν προτάσεις και να δημιουργήσουν προγράμματα. Οι κανόνες αυτοί, γνωστοί με το όνομα *συντακτικοί κανόνες*, αποτελούν το συντακτικό της γλώσσας. Η ακριβής εφαρμογή του συντακτικού της γλώσσας έχει δραματική επίδραση στην αξιοπιστία των προγραμμάτων. Στη βιβλιογραφία αναφέρονται πολλά παραδείγματα συστημάτων που καταστράφηκαν εξαιτίας λαθών που ένα καλύτερο συντακτικό

[1] Ανατρέξτε σε ένα από τα βιβλία της βιβλιογραφίας που αναφέρονται στη C για μία πλήρη λίστα του ASCII κώδικα.

δεν θα είχε επιτρέψει. Μερικά τέτοια παραδείγματα, μπορείτε να βρείτε στο αντίστοιχο κεφάλαιο του βιβλίου [Horowitz '84] ή του [Horowitz '95]. Στα ίδια βιβλία, μπορείτε να ανατρέξετε για περισσότερες πληροφορίες για τους διάφορους τρόπους έκφρασης του συντακτικού μιας γλώσσας προγραμματισμού. Ορισμένοι από αυτούς αναφέρονται στο ειδικό ένθετο πλαίσιο.

Τρόποι περιγραφής συντακτικού γλώσσας

• Backus–Naur–Form (BNF)

Η BNF σημειογραφία αναπτύχθηκε το 1963 για να περιγράψει τη σύνταξη της γλώσσας ALGOL60. Αποτελεί με τις επεκτάσεις της (π.χ. EBNF) μια από τις πλέον δημοφιλείς σημειογραφίες μετασύνταξης για τον προσδιορισμό του συντακτικού των γλωσσών προγραμματισμού. Παρόλα αυτά δεν είναι τεκμηριωμένη ικανοποιητικά στην βιβλιογραφία. Μία τεκμηρίωση της Augumented BNF (επέκτασης της BNF) δίνεται σαν πρότυπο RFC 2234 και μπορείτε να τη βρείτε στο διαδίκτυο στη διεύθυνση http://www.nexor.com/public/rfc/index/cgi–bin/search/form?2234

• Συντακτικό διάγραμμα (syntax graph)

Η μέθοδος είναι ισοδύναμη με την BNF, με σαφείς κανόνες αντιστοίχησης που ορίστηκαν από τον Wirth [Wirth76]. Η χρήση διαγραμμάτων την κάνει περισσότερο κατανοητή από την BNF μορφή.

• Γραμματικές χωρίς συμφραζόμενα (context–free grammars)

Επινοήθηκε από τον Chomsky και χρησιμοποιείται για τη θεωρητική θεμελίωση μιας γλώσσας. [Δες http://ftp.cs.rochester.edu/u/leblanc/csc173/grammars/]

2.3 Το λεξιλόγιο της γλώσσας

Όπως ήδη αναφέραμε, η κάθε γλώσσα σχηματίζει με τους χαρακτήρες του αλφαβήτου της, τις λέξεις ή τα σύμβολα που αποτελούν το λεξιλόγιό της. Οι λέξεις του λεξιλογίου μπορούν να διακριθούν σε επί μέρους κατηγορίες, μεταξύ των οποίων σημαντικότερες είναι οι εξής:

• *δεσμευμένες λέξεις* (reserved words)

• *λέξεις–κλειδιά* (keywords)

• *τελεστές* (operators)

• *αναγνωριστές* (identifiers)

Στο υπόλοιπο του κεφαλαίου, θα αναφερθούμε στις δεσμευμένες λέξεις, τις

λέξεις–κλειδιά και τους αναγνωριστές. Τους τελεστές θα εξετάσουμε σε επόμενο κεφάλαιο. Μέχρι τότε, καλό είναι να γνωρίζουμε, ότι ένας τελεστής είναι ένα σύμβολο ή μία λέξη που αναπαριστά συγκεκριμένη διεργασία, η οποία εκτελείται πάνω σε ένα η περισσότερα δεδομένα. Ο τελεστής *, για παράδειγμα, αναπαριστά τη διεργασία του πολλαπλασιασμού.

Περιγράψτε τις διεργασίες που κατά τη γνώμη σας αναπαριστούν οι τελεστές +, –, *, /, <.

Άσκηση Αυτοαξιολόγησης 2.1

Δώστε τον ορισμό για τον όρο «συντακτικό γλώσσας».

Άσκηση Αυτοαξιολόγησης 2.2

2.4 Δεσμευμένες λέξεις

Οι δεσμευμένες λέξεις (reserved words) είναι λεκτικές μονάδες των οποίων η σημασία καθορίζεται από τους κανόνες της γλώσσας και δεν μπορεί να αλλάξει από το χρήστη. Οι δεσμευμένες λέξεις της C διακρίνονται στις παρακάτω 6 βασικές κατηγορίες:

1. λέξεις–κλειδιά

   λόγω της σημασίας τους αναπτύσσονται στην επόμενη παράγραφο

2. ονόματα συναρτήσεων της βασικής βιβλιοθήκης (runtime function names)

   printf, isdigit, abs κλπ. αποτελούν ονόματα των έτοιμων συναρτήσεων της βασικής βιβλιοθήκης. Αποφεύγετε να χρησιμοποιείτε τα ονόματα αυτά εκτός εάν θέλετε να γράψετε τη δική σας έκδοση για μια συνάρτηση της βιβλιοθήκης.

3. Macro names

   είναι ονόματα που περιέχονται σε αρχεία επικεφαλίδας της βασικής βιβλιοθήκης, για ορισμό μακροεντολών όπως, για παράδειγμα, EOF, INT_MAX.

4. ονόματα τύπων (type names)

   είναι ονόματα τύπων που ορίζει η βασική βιβλιοθήκη σε ορισμένα αρχεία επικεφαλίδας, όπως, για παράδειγμα, `time_t`, `va_list`.

5. ονόματα εντολών προ–επεξεργαστή (preprocessor)

   είναι ονόματα που χρησιμοποιεί ο προεπεξεργαστής της C και έχουν προκαθορισμένη σημασία π.χ., `include`, `define`.

6. ονόματα που αρχίζουν με το χαρακτήρα υπογράμμισης _ και έχουν σαν δεύτερο χαρακτήρα τον ίδιο ή ένα κεφαλαίο γράμμα π.χ. `__DATE__` , `__FILE__`.

Έναν πλήρη κατάλογο των δεσμευμένων λέξεων της C και της κατηγορίας στην οποία η καθεμιά ανήκει, μπορείτε να βρείτε στο [Darnell 1991] ή στο διαδίκτυο στη διεύθυνση http://www.concentric.net/~Brownsta/c–predef.htm#ReservedIdentifiers, κάτω από τον τίτλο «Identifiers NOT To Use in C Programs – The Reserved Identifiers».

2.5 Λέξεις – κλειδιά

Οι λέξεις–κλειδιά (keywords) είναι λεκτικές μονάδες που μόνες τους ή με άλλες λεκτικές μονάδες χαρακτηρίζουν κάποια γλωσσική κατασκευή. Η λέξη–κλειδί `int` της C, για παράδειγμα, αναπαριστά τον ακέραιο τύπο δεδομένων, ενώ η `if` σε συνδυασμό με τη λέξη–κλειδί `else` δημιουργεί τη βασική γλωσσική κατασκευή `if else` για τον έλεγχο της ροής του προγράμματος.

Οι λέξεις–κλειδιά, αν και είναι ένας περιορισμός των γλωσσών, αυξάνουν την αναγνωσιμότητα και αξιοπιστία των προγραμμάτων, ενώ ταυτόχρονα, επιταχύνουν τη διαδικασία της μεταγλώττισης. Ο αριθμός των λέξεων–κλειδιών διαφέρει από γλώσσα σε γλώσσα, αλλά γενικά, παρατηρούμε μια τάση αύξησής του, γεγονός που δημιουργεί προβλήματα στην απομνημόνευση των λέξεων.

Λέξεις–κλειδιά όπως `if`, `else`, `for`, `case`, `while`, `do` έχουν γίνει κοινά αποδεκτές διευκολύνοντας την εκμάθηση των γλωσσών προγραμματισμού. Ο παρακάτω πίνακας δίνει τις λέξεις–κλειδιά της C, τη σημασία τους, όμως, θα γνωρίσουμε στα επόμενα κεφάλαια.

Λέξεις–κλειδιά στην ANSI C

`auto`	`double`	`int`	`struct`	`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`	`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`	`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`	`do`	`if`	`static`	`while`

2.6 Αναγνωριστές

Στην κατηγορία των αναγνωριστών (identifiers) ανήκουν λεκτικές μονάδες (λέξεις) που κατασκευάζει ο προγραμματιστής. Οι λεκτικές αυτές μονάδες χρησιμοποιούνται συνήθως σαν ονόματα που ο προγραμματιστής δίνει σε δικές του κατασκευές όπως μεταβλητές, σταθερές, συναρτήσεις και δικούς του τύπους δεδομένων. Ένα όνομα προσδιορίζει μονοσήμαντα (uniquely identifies) την κατασκευή στην οποία αποδόθηκε, από το σύνολο των κατασκευών του προγράμματος, απ' όπου και το όνομα αναγνωριστής (identifier).

Η δημιουργία ονομάτων στη C διέπεται από ένα σύνολο κανόνων, οι οποίοι είναι ανεξάρτητοι από τον τύπο του αντικειμένου στον οποίο το όνομα αναφέρεται. Τέτοιοι κανόνες είναι:

1. Τα ονόματα μπορεί να περιέχουν γράμματα, αριθμούς και το χαρακτήρα υπογράμμισης, αλλά πρέπει να αρχίζουν οπωσδήποτε από γράμμα ή χαρακτήρα υπογράμμισης.
2. Ένα όνομα δεν πρέπει να είναι ίδιο με μία δεσμευμένη λέξη
3. Η C θεωρεί διαφορετικό τον μικρό από τον κεφαλαίο χαρακτήρα ενός γράμματος, γι' αυτό λέμε ότι η γλώσσα είναι case sensitive.
4. Η C δεν βάζει όριο στο μήκος ενός ονόματος. Παρόλα αυτά, ο κάθε μεταγλωττιστής βάζει το δικό του όριο, με τον περιορισμό από την ANSI C αυτό να είναι τουλάχιστον 31 χαρακτήρες. Ορισμένοι όμως παλιοί μεταγλωττιστές και συνδέτες έχουν όριό τους 8 χαρακτήρες. Σε κάθε περίπτωση, θα πρέπει να γνωρίζετε το όριο του συστήματος ανάπτυξής σας και να μην το υπερβαίνετε.

Η διαδικασία της απόδοσης ονομάτων στα στοιχεία του προγράμματος είναι ένας από τους σημαντικότερους παράγοντες που καθορίζουν το βαθμό αναγνωσιμότητας του προγράμματος. Ένα καλό πρόγραμμα, όχι μόνο λειτουργεί σωστά, αλλά και είναι εύκολο στην ανάγνωση και συντήρησή του. Μην παραβλέπετε σε καμία περίπτωση την αναγνωσιμότητα ενός προγράμματος. Στην αρχή, προσπαθήστε αφιερώνοντας ίσως περισσότερο χρόνο και από αυτόν που απαιτείται για την επίλυση του προβλήματος. Το αποτέλεσμα θα είναι σημαντικό. Με το χρόνο θα μπορείτε να γράφετε ευανάγνωστα προγράμματα χωρίς πρόσθετη προσπάθεια. Να είστε βέβαιοι ότι θα έχετε αποκτήσει ένα πολύ καλό εφόδιο σαν προγραμματιστές.

Στο ειδικό ένθετο πλαίσιο σας δίνω ορισμένους από τους πρακτικούς κανό-

νες που θα σας βοηθήσουν στη δημιουργία ευανάγνωστου προγράμματος. Άλλους θα συναντήσετε σε επόμενα κεφάλαια.

Κανόνες δημιουργίας ευανάγνωστου προγράμματος

- Αποφύγετε ονόματα ενός χαρακτήρα όπως. `i`, `j`, `l`, `m`. Υπάρχουν βέβαια περιπτώσεις, όπως θα δούμε στη συνέχεια, όπου χρησιμοποιούνται ονόματα αυτής της μορφής.
- Χρησιμοποιήστε εκφραστικά ονόματα. Πιο συγκεκριμένα:

  α) ονομάστε τη μεταβλητή η οποία αναπαριστά την ταχύτητα `velocity` και τη μέγιστη τιμή της `max_velocity` ή `maxVelocity`

  β) ονομάστε τη συνάρτηση που εμφανίζει τα λάθη στην οθόνη `display_error` ή `displayError`.

  Για την καλύτερη αναγνωσιμότητα των μεταβλητών που αποτελούνται από δύο ή περισσότερες λέξεις της ομιλουμένης, αποφασίστε αν θα χρησιμοποιείτε σαν διαχωριστικό τον χαρακτήρα υπογράμμισης ή θα γράφετε κεφαλαίο το πρώτο γράμμα των λέξεων μετά την πρώτη. Προσπαθήστε να μην παραβιάζεται τον κανόνα σας.
- Χρησιμοποιήστε μικρά γράμματα για ονόματα μεταβλητών. Θα δούμε στη συνέχεια, ότι χρησιμοποιούμε κεφαλαία γράμματα μόνο για μακροεντολές.

2.7 Συντακτικά – Σημασιολογικά λάθη

Αν αναρωτηθήκατε, ποιος θα ελέγχει τα προγράμματά σας όσον αφορά την τήρηση του συντακτικού της γλώσσας, ευτυχώς για σας αλλά και για μας, τη δουλειά αυτή κάνει αυτόματα και χωρίς να κουράζεται ή να εκνευρίζεται ο μεταγλωττιστής. Έχετε λοιπόν στη διάθεσή σας έναν ακούραστο ελεγκτή, πρόθυμο να εντοπίσει το κάθε συντακτικό σας λάθος, όποτε εσείς το επιθυμείτε και να σας δώσει οδηγίες διόρθωσής του. Αυτό βέβαια, αποτελεί μία από τις πολύ σημαντικές λειτουργίες του μεταγλωττιστή.

Αν ο μεταγλωττιστής κατά τη διαδικασία του ελέγχου αυτού ανιχνεύσει λάθη που οφείλονται σε παραβίαση των συντακτικών κανόνων, τα αναφέρει υπό μορφή λίστας στον προγραμματιστή, ο οποίος είναι υπεύθυνος για την διόρθωσή τους. Τα λάθη αυτά, που είναι γνωστά ως *συντακτικά λάθη*, αναγνωρίζονται στο χρόνο μεταγλώττισης σε αντίθεση με τα *σημασιολογικά* (semantic errors), τα οποία οφείλονται στην κακή απόδοση της λύσης του προβλήματος και αναγνωρίζονται αργότερα στο χρόνο εκτέλεσης. Μπορείτε να ανα-

τρέξετε στην ενότητα 3.9 του βιβλίου [Rojiani 96] για μια πολύ καλή περιγραφή των διάφορων κατηγοριών λαθών.

Ο μεταγλωττιστής περνάει στη διαδικασία παραγωγής σημασιολογικά ισοδύναμου κώδικα μηχανής, μόνο αν όλες οι προτάσεις ενός προγράμματος είναι συντακτικά σωστές. Η διεργασία αυτή, που είναι γνωστή με το όνομα *συντακτική ανάλυση* (parsing), βασίζεται σε πληθώρα σύνθετων αλγορίθμων, οι οποίοι μπορούν να χειριστούν σύνθετες γλωσσικές μορφές [Aho–Ullman 72].

Δραστηριότητα 2.1

Τα δύο τμήματα κώδικα που δίνονται στη συνέχεια εκτελούν την ίδια ακριβώς λειτουργία αλλά έχουν διαφορετικό βαθμό αναγνωσιμότητας. Μελετήστε τα και γράψτε τα συμπεράσματά σας. Στη συνέχεια, ανατρέξτε στο πλαίσιο «κανόνες δημιουργίας ευανάγνωστου προγράμματος» και επιβεβαιώστε τα συμπεράσματά σας.

Κώδικας 1

```
i = 120;
if(i<j)
   func1();
else
   func2();
```

Κώδικας 2

```
velocity = 120;
if(velocity > max_velocity)
   decrease_velocity();
else
   increase_velosity();
```

Δραστηριότητα 2.2

Από την παρακάτω λίστα ονομάτων αναγνωρίστε ποια δεν είναι σύμφωνα με τους κανόνες σύνθεσης ονομάτων της C και καταγράψτε τον κανόνα που παραβιάζεται. Κάντε το ίδιο θεωρώντας την αναγνωσιμότητά τους. Ελέγξτε το αποτέλεσμα της δουλειάς σας με βάση τους κανόνες που παρατέθηκαν στο κεφάλαιο αυτό.

```
J              $amount            find_max#of_lements
5j             set_password       get_words
lname          int                isdigit
_Fname         MaXvElOcItY        get@name
```

Σύνοψη

Στο κεφάλαιο αυτό, εισάγαμε τις βασικές έννοιες του συντακτικού των γλωσσών προγραμματισμού. Κάθε γλώσσα προγραμματισμού, διαθέτει ένα αλφάβητο, με τους χαρακτήρες του οποίου σχηματίζει τις λέξεις ή τα σύμβολα που αποτελούν το λεξιλόγιό της. Οι λέξεις και τα σύμβολα αυτά, γνωστά σαν δεσμευμένες λέξεις, έχουν προκαθορισμένη σημασία και είναι στη διάθεση του προγραμματιστή για να τον βοηθήσουν στη διαδικασία συγγραφής του πηγαίου κώδικα. Επιπλέον, ο προγραμματιστής χρησιμοποιεί τους χαρακτήρες του λεξιλογίου για να δημιουργήσει ονόματα, τα οποία αποδίδει στις δικές του κατασκευές ώστε να αναφέρεται σε αυτές. Στη διαδικασία αυτή, ένα σύνολο από κανόνες οδηγούν στη συγγραφή ευανάγνωστου προγράμματος. Ένα σύνολο από άλλους κανόνες, που είναι γνωστοί ως συντακτικοί κανόνες, καθορίζουν αν μια πρόταση, και κατ' επέκταση το πρόγραμμα, είναι σχηματισμένο σωστά ή όχι.

Βιβλιογραφία κεφαλαίου

[Aho–Ullman'72]

Aho J. A., Ullman *«The theory of Parsing, Translation and Compiling»* Prentice–Hall 1972.

[ANSI '88]

Περισσότερες πληροφορίες για το πρωτότυπο ISO/IEC 9899 μπορείτε να βρείτε στο δια–δίκτυο στη διεύθυνση http://wwwold.dkuug.dk/JTC1/SC22/WG14/.

[Darnell 1991]

Darnell P., Margolis P. *«C: A Software Engineering Approach»*, Springer–Verlag, New York 1991.

[Horowitz '84]

«Βασικές αρχές γλωσσών προγραμματισμού» Εκδόσεις Κλειδάριθμος,1993.

Το βιβλίο είναι μετάφραση της δεύτερης Αμερικάνικης έκδοσης του *«Fundamentals of Programming Languages»* Ellis Horowitz, Computer Science Press, 1984. Αποτελεί μια πολύ καλή και, κατά τα γνωστά μου, μοναδική στην ελληνική γλώσσα πηγή, αναφορικά με τις βασικές αρχές των γλωσσών προγραμματισμού. Το βιβλίο είναι κατάλληλο για ανάγνωση μόνο επιλεκτικών τμημάτων από σπουδαστές που μαθαίνουν την πρώτη τους γλώσσα προγραμματισμού. Είναι όμως απαραίτητο για σπουδαστές που γνωρίζουν και έχουν χρησιμοποιήσει εκτεταμένα μια τουλάχιστον γλώσσα προγραμματισμού.

[Horowitz '95]

Horowitz Ellis, *«Fundamentals of Programming Languages»,* Third edition, Computer Science Press, 1995.

[Rojiani '96]

Rojiani K. B., *«Programming in C with numerical methods for Engineers»*, Prentice–Hall, 1996.

[Unicode]

Το unicode που τυποποιήθηκε σαν ISO 10646 πρότυπο, βασίζει την κωδικοποίηση των χαρακτήρων σε ένα σχήμα 16 bit (αντί 8 bit που χρη-

σιμοποιούσαν τα προηγούμενα), αυξάνοντας με τον τρόπο αυτό κατά πολύ τον αριθμό των χαρακτήρων που μπορεί να αναπαραστήσει. Πληροφορίες για το unicode μπορείτε να βρείτε στο διαδίκτυο στη διεύθυνση http://www.unicode.org/.

[Wirth '76]

Wirth N., *«Algorithms + Data Structures = Programs»,* Prentice–Hall, 1976.

ΚΕΦΑΛΑΙΟ 3

Μεταβλητές – Σταθερές – Τύποι Δεδομένων

Σκοπός

Σκοπός του κεφαλαίου είναι να εισάγει τις πολύ βασικές έννοιες της μεταβλητής και του τύπου δεδομένων, καθώς και τον τρόπο με τον οποίο αυτές χρησιμοποιούνται στη διαδικασία συγγραφής προγράμματος. Παρουσιάζει, επίσης, τους βασικούς τύπους δεδομένων της C και τον τρόπο με τον οποίο αυτοί χρησιμοποιούνται για τη δήλωση των μεταβλητών.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *περιγράψετε πώς μια γλώσσα προγραμματισμού υποστηρίζει την πρόσβαση στην κύρια μνήμη του υπολογιστή είτε για αποθήκευση είτε για ανάκληση πληροφορίας,*
- *να αναφέρετε τουλάχιστον 4 βασικούς τύπους δεδομένων που συνήθως υποστηρίζουν οι διάφορες γλώσσες προγραμματισμού,*
- *δηλώσετε μεταβλητές διαφόρων τύπων και προαιρετικά να δώσετε αρχική τιμή σε αυτές,*
- *αναγνωρίσετε για το κάθε δεδομένο του προβλήματός σας αν μπορείτε να το κατατάξετε σε μία από τις κατηγορίες δεδομένων που υποστηρίζει η γλώσσα που χρησιμοποιείτε ή θα πρέπει να δημιουργήσετε μία νέα κατηγορία.*

Έννοιες–Κλειδιά

- *μεταβλητή*
- *σταθερή*
- *τύποι δεδομένων*
- *δήλωση μεταβλητής*
- *συναθροιστικός τύπος*
- *παραγόμενος τύπος*
- *τύπος χαρακτήρα*
- *τύπος ακεραίου*
- *τύπος πραγματικού*
- *τύπος πίνακα*
- *τύπος δείκτη*
- *αλφαριθμητικό*

Εισαγωγικές Παρατηρήσεις

Ένα από τα βασικότερα πλεονεκτήματα του υπολογιστή είναι η δυνατότητά

του να διαχειρίζεται πληροφορία. Οι γλώσσες προγραμματισμού χρησιμοποιούν τις έννοιες της μεταβλητής και της σταθερής για να επιτρέψουν στον προγραμματιστή να αναφερθεί στα δεδομένα του προβλήματός του ώστε να τα επεξεργαστεί κατάλληλα. Οι διάφορες μορφές που μπορεί να έχουν τα δεδομένα αυτά, καθιστούν απαραίτητη την ταξινόμησή τους σε κατηγορίες για την καλύτερη οργάνωση και διαχείρισή τους. Οι τύποι δεδομένων βοηθούν προς την κατεύθυνση αυτή.

Το κεφάλαιο αυτό χωρίζεται σε δύο ενότητες. Στην πρώτη από αυτές, αναφερόμαστε στον τρόπο με τον οποίο οι γλώσσες προγραμματισμού χειρίζονται (αποθηκεύουν και ανακαλούν) τα δεδομένα. Στη δεύτερη ενότητα, παρουσιάζουμε τους βασικούς τύπους δεδομένων της C και τον τρόπο χειρισμού των μεταβλητών του κάθε τύπου.

Δραστηριότητα 3.1

Θυμάστε ποια είναι η διαδικασία δημιουργίας εκτελέσιμου κώδικα; Περιγράψτε τα βήματα της διαδικασία αυτής (15 το πολύ γραμμές) και συγκρίνετε την απάντησή σας με όσα αναφέρονται στο αντίστοιχο κεφάλαιο.

3.1 Εισαγωγή στις έννοιες της μεταβλητής, της σταθερής και του τύπου δεδομένων

Σκοπός

Σκοπός της ενότητας είναι να περιγράψει πώς οι γλώσσες προγραμματισμού υποστηρίζουν την πρόσβαση στην κύρια μνήμη του υπολογιστή είτε για αποθήκευση είτε για ανάκληση πληροφορίας.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα αυτή θα μπορείτε να:

- *αναφέρετε πώς οι προγραμματιστές τα πρώτα χρόνια των υπολογιστών είχαν πρόσβαση στην κύρια μνήμη του υπολογιστή,*
- *εξηγήσετε γιατί οι γλώσσες προγραμματισμού χρησιμοποιούν την έννοια της μεταβλητής,*
- *περιγράψετε τις δύο διαφορετικές σημασίες που μπορεί να έχει σε μια πρόταση το όνομα μιας μεταβλητής,*
- *περιγράψετε τι κάνει μια δήλωση μεταβλητής,*
- *εξηγήσετε τι είναι οι παραγόμενοι τύποι και γιατί είναι σημαντικό να υποστηρίζονται από μια γλώσσα προγραμματισμού,*
- *δώσετε 3 τουλάχιστον παραδείγματα παραγόμενων τύπων,*
- *δηλώσετε μια ακέραια μεταβλητή και να της αποδώσετε αρχική τιμή,*
- *δηλώνετε μεταβλητές των οποίων η τιμή δεν θέλετε να αλλάζει κατά τη διάρκεια εκτέλεσης του προγράμματος,*
- *αναφέρετε τουλάχιστον 2 συναθροιστικούς τύπους δεδομένων.*

Εισαγωγικές Παρατηρήσεις

Όπως ήδη αναφέραμε, ένα από τα βασικότερα πλεονεκτήματα του υπολογιστή είναι η δυνατότητά του να διαχειρίζεται πληροφορία. Η πληροφορία αυτή, που μπορεί να έχει τη μορφή αριθμητικών δεδομένων, γραμμάτων, ακολουθιών γραμμάτων κλπ., αποθηκεύεται στη μνήμη του υπολογιστή και αποτελεί συνήθως τα δεδομένα εισόδου για το πρόγραμμα. Επιπλέον, κατά τη διάρκεια της διαχείρισης των δεδομένων αυτών, νέα δεδομένα προκύπτουν, τα οποία είναι είτε τελικά αποτελέσματα που ο χρήστης αναμένει είτε ενδιάμε-

σα αποτελέσματα που θα χρησιμοποιηθούν για την παραγωγή των τελικών αποτελεσμάτων.

Ένα πρόγραμμα για να διαχειριστεί τα δεδομένα αυτά (εισόδου, ενδιάμεσα, εξόδου), πρέπει να έχει τρόπο αναφοράς σε αυτά. Πιο συγκεκριμένα, το πρόγραμμα θα πρέπει να έχει τη δυνατότητα να διαβάζει τα δεδομένα αυτά από τη θέση της μνήμης που βρίσκονται αποθηκευμένα, αλλά και να τα τροποποιεί.

Τα πρώτα χρόνια ανάπτυξης προγραμμάτων, ο προγραμματιστής χρησιμοποιούσε έναν αριθμό, που ήταν η ακριβής διεύθυνση της μνήμης όπου ήταν αποθηκευμένη η πληροφορία. Με την αύξηση όμως του μεγέθους και της πολυπλοκότητας των προγραμμάτων, ο τρόπος αυτός διαχείρισης έθετε πολλούς περιορισμούς και ταυτόχρονα, αποτελούσε πηγή πολλών προβλημάτων.

3.1.1 Τι είναι μεταβλητή;

Η έννοια της μεταβλητής ήταν αυτή που έδωσε λύση στο πρόβλημα της αναφοράς σε πληροφορία της κύριας μνήμης του υπολογιστή. Οι γλώσσες προγραμματισμού υποστηρίζουν την πρόσβαση στα δεδομένα με τη χρήση συμβολικών ονομάτων, τα οποία καλούνται μεταβλητές. Το όνομα της μεταβλητής είναι άμεσα συνδεμένο με την ακριβή διεύθυνση της μνήμης όπου είναι αποθηκευμένη η τρέχουσα τιμή της μεταβλητής. Μας επιτρέπει δε, να αναφερόμαστε αφενός μεν στη τιμή χωρίς να είναι απαραίτητο να γνωρίζουμε την ακριβή διεύθυνση της μνήμης που αυτή είναι αποθηκευμένη, αφετέρου δε στη θέση της μνήμης για να τροποποιήσουμε την τρέχουσα τιμή της. Είναι φανερό λοιπόν, ότι το όνομα της μεταβλητής έχει διπλή σημασία και για να γίνει αυτό πιο κατανοητό, ας θεωρήσουμε την μεταβλητή `count` με τιμή 12 και την πρόταση της C:

```
count = count + 1;
```

Η πρόταση, η οποία χρησιμοποιεί τους τελεστές ανάθεσης (=) και πρόσθεσης (+), για να αυξήσει την τιμή της μεταβλητής `count` κατά 1, μπορεί να ερμηνευθεί ως ακολούθως: Το όνομα `count` στο δεξιό μέρος του τελεστή ανάθεσης αναφέρεται στην τρέχουσα τιμή της μεταβλητής, δηλαδή το 12, την τιμή αυτή προσθέτει ο υπολογιστής με το 1 (αυτό του επιβάλει να κάνει ο τελεστής πρόσθεσης) και το αποτέλεσμα 13 το τοποθετεί, όπως του ορίζει ο τελεστής ανάθεσης, στη θέση μνήμης στην οποία αναφέρεται το όνομα `count` στο αριστερό μέρος του τελεστή ανάθεσης.

Ένας πιο αυστηρός ορισμός από τον Horowitz, ορίζει τη μεταβλητή αποτελούμενη από ένα συμβολικό όνομα, ένα σύνολο ιδιοτήτων, μια αναφορά και μία τιμή [Horowitz 95].

Με το επόμενο παράδειγμα, θα προσπαθήσουμε να δείξουμε πώς η έννοια της μεταβλητής εμπλέκεται στην επίλυση ενός προβλήματος. Ας υποθέσουμε πως θέλουμε να υπολογίσουμε την επιφάνεια ενός οικοπέδου σχήματος ορθογωνίου. Το πρόγραμμα που θα αναπτύξουμε έχει σαν δεδομένα εισόδου το πλάτος και το ύψος του οικοπέδου και σαν έξοδο (αποτέλεσμα) το ζητούμενο εμβαδόν του οικοπέδου. Είναι προφανές, ότι η διεργασία που θα εκτελεί το πρόγραμμα είναι ο πολλαπλασιασμός των δύο δεδομένων εισόδου μεταξύ τους. Για την αναπαράσταση των δεδομένων εισόδου και εξόδου χρησιμοποιούμε τρεις μεταβλητές τις οποίες ονομάζουμε `width`, `height` και `area` αντίστοιχα. Με δεδομένο ότι η διεργασία του πολλαπλασιασμού εκτελείται, όπως και στα μαθηματικά, από τον τελεστή `*`, χρησιμοποιούμε την παρακάτω πρόταση για τον υπολογισμό του εμβαδού

```
area = width * height;
```

Η πρόταση οδηγεί τον υπολογιστή να πολλαπλασιάσει την τιμή της μεταβλητής `width` με την τιμή της μεταβλητής `height` και να τοποθετήσει το αποτέλεσμα στη θέση μνήμης στην οποία αναφέρεται το όνομα `area`. Στη συνέχεια, θα δούμε πώς από εκεί εύκολα μπορεί να εμφανιστεί στην οθόνη.

3.1.2 Δήλωση μεταβλητής

Διαφορετικές ιδιότητες έχει μία μεταβλητή η οποία αναπαριστά τη μέση θερμοκρασία ενός δωματίου από άλλη, η οποία αναπαριστά το επίθετο ενός φοιτητή. Οι ιδιότητες μιας μεταβλητής (θυμόσαστε τον ορισμό του Horowitz), για ορισμένες γλώσσες προγραμματισμού ορίζονται στη διάρκεια της μεταγλώττισης και δεν μεταβάλλονται, για δε άλλες, ορίζονται στο χρόνο εκτέλεσης και μπορούν να μεταβληθούν. Μεταξύ των ιδιοτήτων αυτών περιλαμβάνεται το είδος των τιμών που η μεταβλητή μπορεί να πάρει, οι μορφές επεξεργασίας που μπορεί να υποστεί, ο χώρος της μνήμης που απαιτεί για την αποθήκευσή της, κ.λπ.

Η C ανήκει στην πρώτη κατηγορία. Απαιτεί από τον προγραμματιστή να προσδιορίζει τις ιδιότητες κάθε μεταβλητής στη διάρκεια της μεταγλώττισης. Απαιτεί δηλαδή να δηλώνεται μια μεταβλητή πριν από τη χρήση της. Η δήλωση μιας μεταβλητής γνωστοποιεί στο μεταγλωττιστή το όνομα και τις

ιδιότητες της μεταβλητής. Πιο αναλυτικά, μία δήλωση έχει σαν αποτέλεσμα τη σύνδεση του ονόματος της μεταβλητής:

• με τον ανάλογο τύπο δεδομένων, γεγονός που λαμβάνει χώρα στο χρόνο μεταγλώττισης (compile–time),

• με μία θέση μνήμης κατάλληλου μεγέθους, γεγονός που λαμβάνει χώρα στο χρόνο εκτέλεσης (run–time).

Ο προγραμματιστής, συνθέτει το όνομα της μεταβλητής εφαρμόζοντας τους κανόνες σύνθεσης και αναγνωσιμότητας ονομάτων, τους οποίους μελετήσατε στο προηγούμενο κεφάλαιο. Αποδίδει δε τις ιδιότητές της ορίζοντας τον τύπο της.

3.1.3 Τι είναι οι τύποι δεδομένων;

Ένας τύπος δεδομένων είναι ένα σύνολο από αντικείμενα με κοινά χαρακτηριστικά τα οποία, συνήθως, εκφράζονται από ένα σύνολο πράξεων πάνω στα αντικείμενα. Οι σύγχρονες προστακτικές γλώσσες, αφενός μεν περιλαμβάνουν ορισμένους ενσωματωμένους τύπους, αφετέρου δε προσφέρουν ένα μηχανισμό για τον ορισμό νέων τύπων. Η δυνατότητα ορισμού νέων τύπων αυξάνει, όπως θα δούμε αργότερα, όχι μόνο την αναγνωσιμότητα του προγράμματος αλλά και την αξιοπιστία του.

Παραδείγματα ενσωματωμένων τύπων για την C αποτελούν ο χαρακτήρας, ο ακέραιος και ο αριθμός κινητής υποδιαστολής απλής ακρίβειας που αναπαρίστανται από τις λέξεις– κλειδιά `char`, `int` και `float` αντίστοιχα. Παράδειγμα παραγόμενου τύπου αποτελεί η δομή φοιτητής (`struct student`), η οποία ορίζει και ομαδοποιεί τις ιδιότητες των μεταβλητών που είναι απαραίτητες για τη διαχείριση των στοιχείων ενός φοιτητή από ένα πρόγραμμα υποστήριξης Πανεπιστημιακής κοινότητας. Τον τρόπο ορισμού και διαχείρισης των παραγόμενων τύπων θα δούμε σε επόμενο κεφάλαιο.

Οι τύποι δεδομένων διακρίνονται σε βαθμωτούς (scalar) και συναθροιστικούς (aggregate). Στην πρώτη κατηγορία, ανήκουν τύποι των οποίων οι τιμές βρίσκονται κατά μήκος μιας γραμμικής κλίμακας, με κλασσικά παραδείγματα τους τύπους του ακέραιου, του πραγματικού, του χαρακτήρα και του λογικού τύπου (boolean). Στη δεύτερη κατηγορία, ανήκουν τύποι οι οποίοι δημιουργούνται συνδυάζοντας έναν ή περισσότερους βαθμωτούς τύπους. Κλασσικά παραδείγματα συναθροιστικών τύπων είναι ο πίνακας και η εγγραφή (record) για την Pascal, ή δομή (struct) για την C.

3.1.4 Δήλωση μεταβλητών στη C

Η δήλωση μιας μεταβλητής στη γλώσσα C έχει την παρακάτω μορφή:

<όνομα τύπου δεδομένων> <λίστα ονομάτων μεταβλητών>;

ή

<όνομα τύπου δεδομένων> <όνομα μεταβλητής> = <αρχική τιμή μεταβλητής>;

Με την πρώτη μορφή δηλώνουμε, χωρίς να αποδίδουμε αρχική τιμή, μία η περισσότερες μεταβλητές που τις διαχωρίζουμε μεταξύ τους με κόμμα. Για τη δήλωση δύο ακέραιων μεταβλητών με ονόματα `count` και `num` μπορούμε να γράψουμε:

```
int count;      /* η πρόταση δηλώνει την ακέραια μεταβλητή count */

int num;        /* η πρόταση δηλώνει την ακέραια μεταβλητή num */
```

ή χρησιμοποιώντας μία πρόταση

```
int count, num;        /* η πρόταση δηλώνει τις ακέραιες μεταβλητές*/
                       */count και num */
```

Με τη δεύτερη μορφή αποδίδουμε αρχική τιμή στη μεταβλητή, ταυτόχρονα με τη δήλωσή της. Η παρακάτω πρόταση δηλώνει την ακέραια μεταβλητή `num` και της αποδίδει αρχική τιμή 20.

```
int num = 20;
```

3.1.5 Μεταβλητές που δεν αλλάζουν τιμή

Υπάρχουν περιπτώσεις που δε θέλουμε μία μεταβλητή να αλλάξει τιμή, μετά την ανάθεση της αρχικής της τιμής και κατά τη διάρκεια της εκτέλεσης του προγράμματος. Στην περίπτωση αυτή, πριν από το όνομα του τύπου βάζουμε την λέξη κλειδί `const`. Για παράδειγμα:

```
const float pi = 3.14; /* το pi είναι σταθερό, δεν θέλουμε να αλλάξει*/
                       /*τιμή */
```

Ο μεταγλωττιστής εκμεταλλεύεται αυτή την πληροφορία, για να αναφέρει ως μη αποδεκτή οποιαδήποτε πρόταση με την οποία ο προγραμματιστής προσπαθεί, κατά λάθος, να αναθέσει τιμή στη μεταβλητή `pi`, αυξάνοντας έτσι την αξιοπιστία του προγράμματος. Μια απρόσεκτη αλλαγή της τιμής του `pi` από τον προγραμματιστή, θα οδηγούσε σε λάθος αποτέλεσμα όλες τις μετέπειτα εκφράσεις που το χρησιμοποιούν.

3.1.6 Η έννοια της σταθεράς

Η επίλυση πολλών προβλημάτων εμπλέκει τη χρήση αριθμητικών σταθερών. Ο υπολογισμός, για παράδειγμα, της περιμέτρου ενός κύκλου γίνεται σύμφωνα με τον τύπο $l = 2\pi r$, όπου το π είναι η γνωστή σταθερά 3.14. Η πρόταση της C που υπολογίζει την περίμετρο του κύκλου είναι:

```
perimetros = 2 * 3.14 * radius;
```

όπου `perimetros` και `radius` είναι πραγματικές μεταβλητές και οι αριθμοί 2 και 3.14 είναι σταθερές. Οι σταθερές επομένως, είναι τιμές, αριθμητικές ή αλφαριθμητικές όπως θα δούμε στη συνέχεια, που χρησιμοποιούνται άμεσα σε προτάσεις.

Άσκηση Αυτοαξιολόγησης 3.1

Αντιστοιχήστε κάθε έναν από τους παρακάτω τύπους με την κατηγορία ή τις κατηγορίες στις οποίες ανήκει.

χαρακτήρας	ενσωματωμένος (για την C)
ακέραιος	παραγόμενος (για την C)
πραγματικός	
πίνακας	βαθμωτός
ημερομηνία	συναθροιστικός

3.2 Οι τύποι δεδομένων στη γλώσσα C

Σκοπός

Ο σκοπός της ενότητας είναι να παρουσιάσει τους βασικούς τύπους δεδομένων της C, καθώς και τον τρόπο που χρησιμοποιούνται στη δήλωση μεταβλητών. Παρουσιάζει επίσης, τον τρόπο απόδοσης τιμής σε μεταβλητές των βασικών αυτών τύπων από την κύρια είσοδο, αλλά και εμφάνισης των τιμών τους στην κύρια έξοδο.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα, θα μπορείτε να:

- *δηλώσετε για τους 4 βασικούς τύπους της C τουλάχιστον δυο μεταβλητές και προαιρετικά να τις αποδώσετε αρχικές τιμές,*
- *γράψετε προτάσεις για την απόδοση τιμής από την κύρια είσοδο σε μεταβλητές των τεσσάρων βασικών τύπων,*

- *γράψετε προτάσεις για την εμφάνιση της τιμής μεταβλητών των τεσσάρων βασικών τύπων στην κύρια έξοδο,*
- *αναφέρετε τον τρόπο με τον οποίο η C χειρίζεται τους μη εκτυπούμενους χαρακτήρες,*
- *αναπτύξετε ένα μικρό πρόγραμμα με απλές λειτουργίες.*

Εισαγωγικές Παρατηρήσεις

Η C προσφέρει ένα μικρό αλλά χρήσιμο σύνολο τύπων δεδομένων. Στην κατηγορία των βαθμωτών τύπων έχει τους αριθμούς, ακέραιους (`int`) και πραγματικούς (`float` και `double`), το χαρακτήρα (`char`), τους δείκτες (`pointers`) και τον απαριθμητικό (`enum`) τύπο. Στην κατηγορία των συναθροιστικών έχει τους πίνακες, τις δομές (`struct`) και τις ενώσεις (`union`). Στη συνέχεια, γίνεται μια σύντομη αναφορά στους βασικούς τύπους της C που είναι οι `char`, `int`, `float`, και `double`. Τους υπόλοιπους θα αφήσουμε για επόμενο κεφάλαιο.

*Στην ενότητα αυτή, θα σας ζητηθεί να αναπτύξετε και **το πρώτο δικό σας πρόγραμμα σε C**. Αυτό σημαίνει ότι, πρέπει να εκτελέσετε τα βήματα της φάσης της υλοποίησης και να παράγετε τον εκτελέσιμο κώδικα, τον οποίο πρέπει να εκτελέσετε για να διαπιστώσετε αν πράγματι το πρόγραμμά σας κάνει τις λειτουργίες που εσείς θέλετε. Αν όχι, εντοπίστε τα λάθη, διορθώστε τα και επαναλάβετε την διαδικασία έως ότου το πρόγραμμά σας ικανοποιήσει τις απαιτήσεις του προβλήματος.*

Σε ορισμένα σημεία, γίνεται αναφορά σε βιβλιογραφία χωρίς να ορίζεται συγκεκριμένο βιβλίο. Αυτό συμβαίνει γιατί το θέμα για το οποίο παραπέμπεστε στη βιβλιογραφία καλύπτεται από κάθε σχεδόν βιβλίο που αναφέρεται στη C. Αυτό, σας δίνει την ελευθερία να ανατρέξετε στο πιο άμεσα διαθέσιμο. Μία λίστα από βιβλία που ανήκουν στην κατηγορία αυτή, μπορείτε να βρείτε στο τέλος του κεφαλαίου, κάτω από τον τίτλο «Γενική Βιβλιογραφία».

3.2.1 Τύπος χαρακτήρα

Ο τύπος χαρακτήρα, που δηλώνεται με τη λέξη–κλειδί **`char`**, χρησιμοποιείται για να αναπαραστήσει απλούς χαρακτήρες του αλφαβήτου της γλώσσας. Μια σταθερά τύπου char εμφανίζεται στον πηγαίο κώδικα ανάμεσα στα απλά εισαγωγικά. Παραδείγματα σταθερών τύπου χαρακτήρα είναι: `'C'`, `'2'`, `'*'`, `')'`.

ΔΗΛΩΣΗ

Για τη δήλωση μιας μεταβλητής τύπου χαρακτήρα ακολουθείται ο γενικός

κανόνας δήλωσης μεταβλητής. Έτσι, για να δηλώσουμε μια μεταβλητή χαρακτήρα με όνομα `choice` και αρχική τιμή Α γράφουμε την παρακάτω πρόταση:

```
char choice = 'A';
```

ΕΚΤΥΠΩΣΗ

Η εκτύπωση της μεταβλητής χαρακτήρα γίνεται με την συνάρτηση `printf` της βασικής βιβλιοθήκης χρησιμοποιώντας τον προσδιοριστή `%c`.

```
printf("ο χαρακτήρας είναι %c\n", choice);
```

Αν αντί του `%c` χρησιμοποιήσουμε τον `%d`, η printf θα εμφανίσει τον ASCII κωδικό του χαρακτήρα. Η πρόταση

```
printf("ο ASCII κώδικας του χαρακτήρα %c είναι %d\n", choice,
      choice);
```

θα τυπώσει στην οθόνη

ο ASCII κώδικας του χαρακτήρα A είναι 65

ΕΙΣΑΓΩΓΗ

Τους ίδιους προσδιοριστές με την `printf` χρησιμοποιεί και η `scanf` για την εισαγωγή τιμής από την κύρια είσοδο (standard input) σε μεταβλητή τύπου χαρακτήρα. Η πρόταση

```
scanf("%c",&ch);
```

διαβάζει από την κύρια είσοδο ένα χαρακτήρα και τον αποδίδει στην μεταβλητή `ch`. Προσέξτε τη χρήση του χαρακτήρα `&` πριν από το όνομα της μεταβλητής. Ο χαρακτήρας `&` είναι ο τελεστής διεύθυνσης, ο οποίος εφαρμοζόμενος σε μία μεταβλητή, δίνει τη διεύθυνση της μεταβλητής. Η `scanf` έχει οριστεί ώστε να δέχεται σαν όρισμα τη διεύθυνση στην οποία θα αποθηκεύσει την τιμή που δέχεται από την κύρια είσοδο. Αν αυτά τα τελευταία σας φαίνονται λίγο περίεργα, δεν έχετε άδικο. Αφήστε τα προς το παρόν. Θα επανέλθουμε αργότερα.

ΑΠΟΘΗΚΕΥΣΗ

Ο μεταγλωττιστής χρειάζεται ένα byte[1] μνήμης, για την αποθήκευση της τιμής μιας μεταβλητής χαρακτήρα στην πράξη όμως δεσμεύεται μια λέξη (word). Στο

[1] Αποτελεί τη συνηθισμένη μονάδα αποθήκευσης που, σύμφωνα με το ANSI πρότυπο, αποτελείται από 8bits.

σχήμα 3.1 φαίνονται παραστατικά, για τον χαρακτήρα A, οι διαδικασίες της αποθήκευσης καθώς και της ανάκλησης για εμφάνιση με την `printf`. Η τιμή της μεταβλητής, μετατρέπεται σε έναν ακέραιο[1] (ενέργεια 1 σχήματος 3.1), ο οποίος αποθηκεύεται (ενέργεια 2). Στην περίπτωση ανάκλησης της τιμής, εκτελείται η αντίστροφη διεργασία. Ο αριθμός μετατρέπεται σε χαρακτήρα λόγω του προσδιοριστή %c και ο χαρακτήρας στη συνέχεια τυπώνεται, για παράδειγμα, στην οθόνη (ενέργεια 3), ή τυπώνεται σαν δεκαδικός λόγω του προσδιοριστή `%d` (ενέργεια 4). Σε κάθε περίπτωση, ο μεταγλωττιστής είναι υπεύθυνος για το ότι ο υπολογιστής διαχειρίζεται τα bits και bytes σύμφωνα με τους δηλωθέντες τύπους.

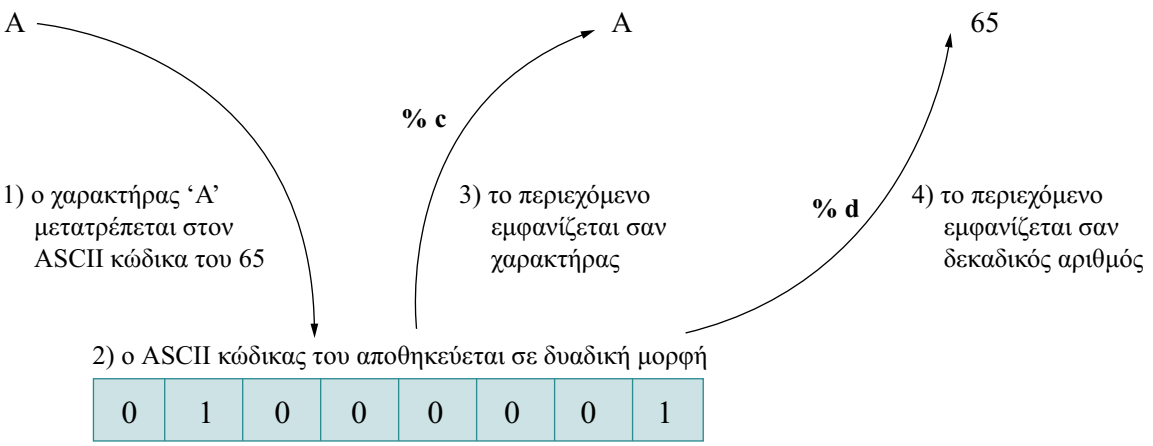
ΜΗ ΕΚΤΥΠΟΥΜΕΝΟΙ ΧΑΡΑΚΤΗΡΕΣ

Ενδιαφέρον είναι να δούμε τον τρόπο με τον οποίο αναπαριστούμε τις σταθερές τύπου χαρακτήρα, νέα–γραμμή (new–line) και στηλοθέτη (tab). Οι δύο χαρακτήρες ανήκουν στην κατηγορία των μη εκτυπούμενων χαρακτήρων (δες βιβλία αναφοράς), τους οποίους η C αναπαριστά με τις ακολουθίες διαφυγής (escape sequences) `'\n'` και `'\t'`, αντίστοιχα. Η παρακάτω πρόταση δίνεται σαν παράδειγμα χρήσης χαρακτήρων διαφυγής.

```
printf("Write, \"a \\ is a backslash. \"\n");
```

Η πρόταση θα εμφανίσει την κύρια έξοδο

Write, “a \ is a backslash.”



Σχήμα 3.1

Αποθήκευση και ανάκληση ASCII χαρακτήρα.

[1] Κάθε χαρακτήρας έχει ένα μοναδικό αριθμητικό ακέραιο κωδικό που είναι ο ASCII κώδικάς του.

3.2.2 Το πρώτο πρόγραμμά σας

Ήρθε η ώρα να γράψετε το πρώτο σας πρόγραμμα στη C. Ας υποθέσουμε ότι θέλουμε ένα πρόγραμμα που να ζητά από τον χρήστη να του δώσει ένα χαρακτήρα και, στη συνέχεια, να τυπώνει τον χαρακτήρα και τον επόμενό του, καθώς και τους ASCII κωδικούς τους.

Στο πρώτο αυτό πρόγραμμα, θα σας δώσω πολλές οδηγίες, θα το αναπτύξουμε μαζί. Σαν πρώτο βήμα, θα πρέπει να κατανοήσετε πλήρως το τι σας ζητάει το πρόβλημα. Στη συνέχεια, θα πρέπει να περιγράψετε με δομημένα ελληνικά τα βήματα που πρέπει να ακολουθήσει μια οντότητα για να εκτελέσει τα ζητούμενα από το πρόβλημα. Για να δούμε μια τέτοια περιγραφή.

1. Ζήτα από το χρήστη ένα χαρακτήρα.
2. Πάρε από το χρήστη το χαρακτήρα.
3. Τύπωσε το χαρακτήρα και τον ASCII κωδικό του.
4. Βρες τον επόμενο χαρακτήρα.
5. Τύπωσέ τον μαζί με τον κωδικό του.

Η παραπάνω περιγραφή αποτελεί τον αλγόριθμο που θα ακολουθήσουμε για την επίλυση του προβλήματος. Τον αλγόριθμο αυτόν πρέπει να εκφράσετε με την γλώσσα C ώστε, στη συνέχεια, να μπορεί να μεταγλωττιστεί και να εκτελεστεί από την οντότητα «Υπολογιστής». Προχωρούμε λοιπόν στην έκφραση μιας προς μιας των προτάσεων του αλγόριθμου σε C.

Πώς μια οντότητα θα μπορούσε να υλοποιήσει την ενέργεια 1; Με ομιλία θα ήταν η καλύτερη ίσως επιλογή (ήδη οι υπολογιστές διαθέτουν ήχο όπως όλοι σας γνωρίζετε), δύσκολη επιλογή όμως για το στάδιο στο οποίο είμαστε. Ας αρκεστούμε προς το παρόν στην εκτύπωση ενός μηνύματος στην οθόνη, το οποίο θα ζητά από το χρήστη την εισαγωγή ενός χαρακτήρα. Η λειτουργία αυτή υλοποιείται από τη συνάρτηση `printf` της βασικής βιβλιοθήκης την οποία έχουμε ήδη χρησιμοποιήσει. Έχουμε λοιπόν:

```
printf("Δώσε ένα χαρακτήρα:\t");
```

Χρησιμοποιήσαμε την ακολουθία χαρακτήρων \t για να μετακινήσουμε το δρομέα στον επόμενο στηλοθέτη.

Η ενέργεια 2 προϋποθέτει την ύπαρξη μιας θέσης στην κύρια μνήμη του υπολογιστή για την αποθήκευση του χαρακτήρα που ο χρήστης θα δώσει. Θα δηλώσουμε, (συμφωνείτε, υποθέτω;) μια μεταβλητή τύπου χαρακτήρα

```
char ch;
```

Στη συνέχεια, χρησιμοποιούμε τη συνάρτηση `scanf` για να πάρουμε από την κύρια είσοδο το χαρακτήρα του χρήστη.

```
scanf("%c",&ch);       /* Η scanf δίνει τον χαρακτήρα σαν τιμή στην*/
                       */μεταβλητή ch. */
```

Τη λειτουργία 3 γνωρίζετε ήδη ότι θα την εκφράσουμε όπως παρακάτω:

```
printf("ο ASCII κώδικας του χαρακτήρα %c είναι %d\n", ch, ch);
```

Για να εκφράσουμε την ενέργεια 4 θα πρέπει να γνωρίζουμε ότι η C χειρίζεται τους χαρακτήρες σαν ακέραιους. Έτσι, έχει νόημα να γράψουμε:

```
next_ch = ch + 1;
```

όπου βέβαια, η `next _ch` θα πρέπει να έχει δηλωθεί σαν μεταβλητή χαρακτήρα.

Αφήνω την ενέργεια 5 για σας. Ολοκληρώνοντας, το πρόγραμμά μας θα έχει την παρακάτω μορφή:

```
/* το πρόγραμμα διαβάζει ένα χαρακτήρα και τυπώνει τον χαρακτηρα*/
και τον επόμενο του καθώς και τους ASCII κωδικούς τους */
#include <stdio.h> /* εντολή προεπεξεργαστή για συμπερίληψη του*/
                   /* αρχείου stdio.h */
main()
{
char ch, next_ch;
printf("Δώσε ένα χαρακτήρα:\t");
scanf("%c",&ch);
printf("ο ASCII κώδικας του χαρακτήρα %c είναι %d\n", ch, ch);
next_ch = ch + 1;      /* βρίσκει τον επόμενο χαρακτήρα */
printf("ο ASCII κώδικας του χαρακτήρα %c είναι %d\n", next_ch,
        next_ch);
}
```

Όπως παρατηρείτε, χρησιμοποιώ τα σύμβολα /* και */ για να παρεμβάλω σχόλια στο πρόγραμμά μου. Η παρεμβολή σχολίων γίνεται πάντα με στόχο την βελτίωση της αναγνωσιμότητάς του. Η λέξη–κλειδί `main`, ακολουθούμενη από δύο παρενθέσεις, δηλώνει το σημείο από το οποίο ο υπολογιστής θα αρχίσει την εκτέλεση του προγράμματός μου. Το σύνολο των προτάσεων που απαρτίζουν το πρόγραμμά μου, περικλείεται μεταξύ δύο αγκυλών, μια αριστερή { για δήλωση αρχής και μια δεξιά } για δήλωση τέλους.

Δραστηριότητα 3.2

Δημιουργήστε για το παραπάνω πρόγραμμα τον εκτελέσιμο κώδικα και ελέγξτε την λειτουργία του προγράμματος. Στη συνέχεια, θεωρήστε την εναλλακτική πρόταση `printf("ο ASCII κώδικας του χαρακτήρα %c είναι %d\n", ch+1, ch+1);` για την εκτέλεση των ενεργειών 4 και 5, που θα μπορούσαν να είχαν εκφραστεί από μια ενέργεια σαν την παρακάτω

4'. Τύπωσε τον επόμενο χαρακτήρα και τον κωδικό του.

Δραστηριότητα 3.3

Ανατρέξτε στη βιβλιογραφία για να δείτε τον τρόπο αναπαράστασης των μη εκτυπούμενων χαρακτήρων. Αναπτύξτε ένα πρόγραμμα που χρησιμοποιεί όσο το δυνατό περισσότερους από τους χαρακτήρες αυτούς. Μία δική μας έκδοση θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 3.4

Αναπτύξτε ένα πρόγραμμα το οποίο θα ζητά από το χρήστη έναν αριθμό από 66 έως 90. Στη συνέχεια, θα τυπώνει τρεις διαδοχικούς χαρακτήρες με τους αντίστοιχους ASCII κωδικούς τους, με τον ενδιάμεσο χαρακτήρα να είναι αυτός που έχει σαν ASCII κωδικό τον αριθμό που έδωσε ο χρήστης. Τη δική μας άποψη θα βρείτε στο τέλος του βιβλίου.

3.2.3 Τύπος ακεραίου

Ο τύπος ακεραίου, που δηλώνεται με τη λέξη–κλειδί `int`, χρησιμοποιείται για να αναπαραστήσει ακέραιους αριθμούς. Οι ακέραιοι μπορεί να είναι θετικοί ή αρνητικοί, η δε περιοχή των τιμών που μπορούν να πάρουν εξαρτάται από την αρχιτεκτονική του μηχανήματος, επειδή για την αποθήκευσή τους χρησιμοποιούνται συνήθως τόσα bits όσο είναι το μήκος της λέξης του συγκεκριμένου επεξεργαστή. Έτσι, για έναν υπολογιστή με λέξη 16 bit, η περιοχή τιμών του τύπου int είναι από – 32767 έως +32768 για προσημασμένους ακεραίους και από 0 έως 65535 για απρόσημους. Ο προσδιοριστής `long` χρησιμοποιείται πριν από τη λέξη int για να προσδιορίσει ακέραιο με μεγαλύτερο εύρος τιμών. Αντίστοιχα, ο προσδιοριστής ***unsigned*** χρησιμοποιείται στη δήλωση πριν από τη λέξη `int` για να χαρακτηρίσει τη μεταβλητή ως απρόσημη.

```
unsigned int count; /* δηλώνει τη μεταβλητή count σαν απρόσημο ακέραιο. */
```

Για την **εκτύπωση** ακεραίων, η συνάρτηση `printf` της βασικής βιβλιοθήκης δέχεται ένα σύνολο από προσδιοριστές μορφής εμφάνισης. Οι προσδιοριστές `%d`, `%x` και `%o` χρησιμοποιούνται για εμφάνιση σε δεκαδική, δεκαεξαδική και οκταδική μορφή, αντίστοιχα. Η πρόταση

```
printf("dec=%d;, octal=%o;,hex=%x\n", num, num, num);
```

τυπώνει την τιμή της ακέραιας μεταβλητής `num` σε δεκαδική, οκταδική και δεκαεξαδική μορφή. Οι προσδιοριστές `l`, `h` και `u` τοποθετούνται πριν τους παραπάνω για να πετύχουμε εμφάνιση `long`, `short` και `unsigned`, αντίστοιχα. Ανατρέξτε στη βιβλιογραφία για περισσότερες πληροφορίες.

Προσοχή: χρησιμοποιώντας λάθος προσδιοριστή έχουμε απρόβλεπτα αποτελέσματα.

Η **εισαγωγή** γίνεται όπως και στην περίπτωση της μεταβλητής χαρακτήρα με τη διαφορά ότι χρησιμοποιείται ο προσδιοριστής `%d`. Η πρόταση

```
scanf("%d",&num);
```

διαβάζει από την κύρια είσοδο σε δεκαδική μορφή και αποδίδει την τιμή στην ακέραια μεταβλητή num.

ΑΚΕΡΑΙΑ ΣΤΑΘΕΡΑ

Όταν γράφουμε έναν αριθμό στον πηγαίο κώδικα χωρίς δεκαδικό ή εκθετικό μέρος, ο μεταγλωττιστής τον χειρίζεται σαν ακέραια σταθερά. Η σταθερά 245 αποθηκεύεται σαν int, ενώ η σταθερά 100.000 αποθηκεύεται σαν `long`. Για ελεγχόμενη αποθήκευση χρησιμοποιούμε τους προσδιοριστές l ή L μετά τον αριθμό. Η σταθερά 8965L αναγκάζει τον μεταγλωττιστή να δεσμεύσει χώρο για `long int`.

Ανατρέξτε στη βιβλιογραφία για περισσότερες πληροφορίες σχετικά με τους προσδιοριστές και τους τρόπους εμφάνισης των ακέραιων μεταβλητών και των ακέραιων σταθερών.

Δραστηριότητα 3.5

Αναπτύξτε ένα πρόγραμμα το οποίο θα ζητά από το χρήστη δύο ακέραιους, θα υπολογίζει το άθροισμά τους και, στη συνέχεια, θα τυπώνει

“οκταδική μορφή δεκαδική μορφή δεκαεξαδική μορφή”

και στην από κάτω γραμμή το άθροισμα που υπολόγισε στις αντίστοιχες μορφές. Τη δική μας άποψη θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 3.6

Ανατρέξτε στο αρχείο limits.h που συνοδεύει το μεταγλωττιστή σας για να δείτε τα όρια των διαφόρων τύπων ακεραίων: `short`, `long`, `signed`, `unsigned`. Δημιουργήστε ένα πρόγραμμα στο οποίο θα δηλώσετε μεταβλητές των παραπάνω τύπων, θα ζητήσετε αντίστοιχες τιμές από το χρήστη και, στη συνέχεια, θα τυπώσετε στην οθόνη τις τιμές αυξημένες κατά ένα. Τι συμβαίνει αν ο χρήστης δώσει τη μέγιστη επιτρεπόμενη τιμή; Τα σχόλιά μας θα βρείτε στο τέλος του βιβλίου.

3.2.4 Τύποι πραγματικών αριθμών

Η C για περισσότερη ευελιξία, διαθέτει 2 τύπους για αναπαράσταση πραγματικών αριθμών[1]. Τον τύπο `float` για αριθμούς κινητής υποδιαστολής απλής ακρίβειας και τον τύπο **double** για τους αριθμούς κινητής υποδιαστολής διπλής ακρίβειας. Η πρόταση

```
float plank = 6.63e–34;
```

δηλώνει τη μεταβλητή `plank` σαν απλής ακρίβειας και της δίνει αρχική τιμή 6.63e–34.

Η χρήση του προσδιοριστή `long` πριν από τον τύπο `double` χρησιμοποιείται για δήλωση μεταβλητής κινητής υποδιαστολής εκτεταμένης ακρίβειας, π.χ.,

```
long double plank;
```

Ο `float`, αν και εξαρτάται από την υλοποίηση, έχει ακρίβεια τουλάχιστον 6 ή 7 δεκαδικά ψηφία, ενώ ο `double` έχει συνήθως διπλάσια ακρίβεια με 14 ή 15 δεκαδικά ψηφία. Μπορείτε να βρείτε τα ακριβή όρια των περιοχών τιμών των πραγματικών αριθμών στο αρχείο επικεφαλίδας `float.h` της βασικής βιβλιοθήκης.

Για την **εκτύπωση** πραγματικών αριθμών, η συνάρτηση `printf` της βασικής βιβλιοθήκης αναγνωρίζει τους προσδιοριστές `%f`, `%e` και `%g`. Ο `%f` έχει σαν αποτέλεσμα την εμφάνιση του αριθμού σε fixed point μορφή, ο `%e` σε εκθετική και ο `%g` αναθέτει στο σύστημα να επιλέξει μεταξύ των δύο προηγούμενων, με προτεραιότητα στη μορφή με το μικρότερο μέγεθος. Για παράδειγμα, αν `float val = 32000.0;` τότε η πρόταση

```
printf("%f μπορεί να γραφεί σαν %e και σαν %g\n", val, val, val);
```

[1] Όπως γνωρίζετε, πραγματικοί είναι οι αριθμοί που διαθέτουν κλασματικό μέρος.

έχει σαν έξοδο

32000.000000 μπορεί να γραφεί σαν 3.200000e+004 και σαν 32000

Ανατρέξτε στη βιβλιογραφία για να δείτε πώς ειδικοί προσδιοριστές επιτρέπουν για τη συνάρτηση `printf`, τον έλεγχο του πλάτους των πεδίων καθώς και του αριθμού των δεκαδικών ψηφίων.

ΠΡΑΓΜΑΤΙΚΕΣ ΣΤΑΘΕΡΕΣ

Πραγματικοί αριθμοί, όπως οι:

0.12 45.68 9e–05 24e09 0.0034e–08

όταν εμφανίζονται στον πηγαίο κώδικα, αποτελούν τις πραγματικές σταθερές. Θεωρούνται από το μεταγλωττιστή σαν `double`, ανεξάρτητα από την ακρίβειά τους, και δεσμεύουν αντίστοιχο χώρο.

ΑΠΟΘΗΚΕΥΣΗ

Η ANSI C δεν ορίζει το χώρο που απαιτείται για αποθήκευση πραγματικής μεταβλητής, με αποτέλεσμα να δημιουργούνται προβλήματα φορητότητας (portability). Σαν συνηθισμένα μεγέθη αναφέρονται, για μεν τους `float`, τα 32 bits από τα οποία τα 8 χρησιμοποιούνται για εκθέτη και πρόσημο και τα υπόλοιπα 24 για βάση, για δε τους `double`, τα 64 bits, με τα επιπλέον 32 bits να χρησιμοποιούνται για αύξηση της ακρίβειας της βάσης.

Άσκηση Αυτοαξιολόγησης 3.2

Αντιστοιχήστε τους παρακάτω προσδιοριστές με τους αντίστοιχους τύπους

%c	δεκαδικός ακέραιος
%f	κινητής υποδιαστολής
%s	
%o	οκταδικός ακέραιος
%x	χαρακτήρας
%g	
%d	αλφαριθμητικό
%e	δεκαεξαδικός ακέραιος

Δραστηριότητα 3.7

Ένας *απαριθμητικός τύπος* είναι ένας τύπος δεδομένων, το πεδίο τιμών του οποίου δίνεται σε μία λίστα, και οι μόνες πράξεις που εφαρμόζονται σε αυτόν είναι η ισότητα και η ανάθεση. Η Pascal ήταν μια από τις πρώτες γλώσσες που έδωσε στον προγραμματιστή τη δυνατότητα να ορίζει δικούς του απαριθμητικούς τύπους. Μελετήστε την ενότητα 2 του πέμπτου κεφαλαίου του βιβλίου «Βασικές Αρχές Γλωσσών Προγραμματισμού» (Κλειδάριθμος 93). Στη συνέχεια, μελετήστε το τέλος της ενότητας 2.3 του βιβλίου «Η γλώσσα προγραμματισμού C» (δεύτερη έκδοση, Κλειδάριθμος, 1990) όπου περιγράφει τον τρόπο με τον οποίο η C υποστηρίζει τον απαριθμητικό τύπο. Εντοπίστε διαφορές στην αντιμετώπιση.

Σύνοψη

Στο κεφάλαιο αυτό, παρουσιάσαμε τον τρόπο με τον οποίο ο προγραμματιστής αναπαριστά τα δεδομένα του προβλήματός του. Αναφέραμε πώς χρησιμοποιεί την έννοια της μεταβλητής σε συνδυασμό με τους τύπους δεδομένων για να δεσμεύσει χώρο για την αποθήκευση και παραπέρα διαχείρισή τους. Είδαμε ότι οι σύγχρονες προστακτικές γλώσσες περιλαμβάνουν ορισμένους ενσωματωμένους τύπους, αλλά το σημαντικότερο, προσφέρουν ειδικό μηχανισμό για τον ορισμό νέων τύπων. Η δυνατότητα αυτή αυξάνει, όχι μόνο την αναγνωσιμότητα, αλλά και την αξιοπιστία του προγράμματος.

Αναφέραμε τέλος, τους βασικούς τύπους της C, το χαρακτήρα, τον ακέραιο, και τον κινητής υποδιαστολής απλής και διπλής ακρίβειας Είδαμε πώς δηλώνονται, εισάγονται, αποθηκεύονται και εκτυπώνονται μεταβλητές αυτών των τύπων και γράψαμε τα πρώτα μας απλά προγράμματα.

Βιβλιογραφία κεφαλαίου

[Horowitz '95]

Horowitz Ellis, *«Fundamentals of Programming Languages»*, Third edition, Computer Science Press, 1995.

Γενική Βιβλιογραφία

[Darnell, 1991]

Darnell P., Margolis P. *«C: A Software Engineering Approach»,* Springer–Verlag, New York, 1991.

[Horowitz '84]

«Βασικές αρχές γλωσσών προγραμματισμού» Εκδόσεις Κλειδάριθμος,1993.

[Kernighan '88]

Η δεύτερη έκδοση του βιβλίου *«The C Programming Language»* κυκλοφόρησε το 1988 καλύπτοντας πλέον την ANSI C. Ελληνική έκδοση, Κλειδάριθμος,1990.

[Rojiani '96]

Rojiani K. B., *«Programming in C with numerical methods for Engineers»*, Prentice–Hall, 1996.

[Waite '90]

Waite Mitchell, Prata Stephen *«New C Primer Plus»* Waite Group Inc., 1990.

ΚΕΦΑΛΑΙΟ 4

Πίνακες – Δείκτες

Σκοπός

Σκοπός του κεφαλαίου είναι να εισάγει τις βασικές έννοιες των πινάκων και των δεικτών στη C, καθώς και τον τρόπο με τον οποίο οι δύο αυτοί τύποι χρησιμοποιούνται για τη δήλωση μεταβλητών. Ιδιαίτερη έμφαση δίνεται στο χειρισμό των αλφαριθμητικών, ως πινάκων χαρακτήρων.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *αναγνωρίζετε πότε είναι απαραίτητη η χρήση του τύπου του πίνακα,*
- *δώσετε παραδείγματα δήλωσης πινάκων και απόδοσης αρχικών τιμών,*
- *αναφέρεστε σε συγκεκριμένο στοιχείο πίνακα,*
- *δηλώνετε και δίνετε αρχική τιμή σε αλφαριθμητικά,*
- *δώσετε τον ορισμό του τύπου του δείκτη,*
- *δηλώνετε μεταβλητές δείκτη και να αποδίδετε σε αυτές αρχικές τιμές,*
- *χρησιμοποιείτε σωστά τους μοναδιαίους τελεστές & και *.*

Έννοιες κλειδιά

- *τύπος πίνακα*
- *πολυδιάστατος πίνακας*
- *αλφαριθμητικό*
- *τύπος δείκτη*
- *τελεστής περιεχομένου **
- *τελεστής διεύθυνσης &*

Εισαγωγικές Παρατηρήσεις

Το κεφάλαιο αυτό χωρίζεται σε δύο ενότητες. Στην πρώτη ενότητα παρουσιάζεται ο συναθροιστικός τύπος του πίνακα, ενώ στη δεύτερη ο τύπος του δείκτη. Οι δύο αυτοί τύποι που έχουν, όπως θα δούμε στη συνέχεια, άμεση σχέση μεταξύ τους, αποτελούν ένα από τα πιο ισχυρά πλεονεκτήματα της C. Η άμεση αντιστοίχηση που υπάρχει μεταξύ τους έχει σαν αποτέλεσμα, όποια λειτουργία γίνεται με πίνακες, να μπορεί να γίνει και με δείκτες. Είμαστε δηλαδή σε θέση να αναφερόμαστε στα ίδια δεδομένα εναλλακτικά και με τους δύο τρόπους.

Ο τύπος του πίνακα χρησιμοποιείται σε μεγάλη έκταση από τις γλώσσες προστακτικού προγραμματισμού και ιδιαίτερα από την C, η οποία χειρίζεται τα αλφαριθμητικά σαν πίνακες. Θα πρέπει, συνεπώς, να δώσετε ιδιαίτερη προσοχή στην πρώτη ενότητα. Αντίθετα, οι δείκτες αποτελούν, κατά γενική ομολογία, όχι μόνο μία από τις δυσκολότερες έννοιες της C, αλλά και αιτία δύσκολων στον εντοπισμό σφαλμάτων. Ζώντας από κοντά χρόνια τώρα τη δυσκολία που όλοι σχεδόν οι σπουδαστές αντιμετωπίζουν στο θέμα αυτό, θα προσπαθήσουμε να δώσουμε στη δεύτερη ενότητα, όσο πιο απλά γίνεται, μόνο τις πολύ βασικές έννοιες. Παρόλα αυτά, οποιαδήποτε δυσκολία στην κατανόηση είναι φυσιολογική και μην σας απογοητεύσει, θα επανέλθουμε στο θέμα και αργότερα. Εξάλλου, λίγοι είναι οι προγραμματιστές που χειρίζονται πολύ καλά τους δείκτες.

4.1 Τύπος πίνακα

Ο πίνακας είναι μια συλλογή μεταβλητών ιδίου τύπου, οι οποίες είναι αποθηκευμένες σε διαδοχικές θέσεις μνήμης. Χρησιμοποιείται για την αποθήκευση και διαχείριση μεγάλων ποσοτήτων δεδομένων που σχετίζονται μεταξύ τους και είναι κοινού τύπου. Είναι μαζί με τους δείκτες από τα πλέον ισχυρά στοιχεία της C. Θα χρησιμοποιήσουμε το παράδειγμα 1 για να εισάγουμε την έννοια του πίνακα.

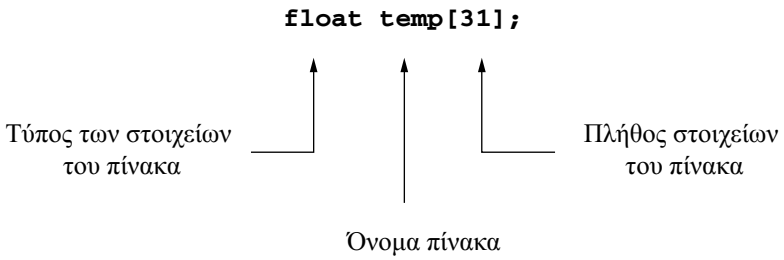
Παράδειγμα 4.1

> Θέλουμε να διαχειριστούμε τις μέσες ημερήσιες θερμοκρασίες ενός μήνα. Θέλουμε, για παράδειγμα, να βρούμε τη μέγιστη, την ελάχιστη και τη μέση μηνιαία θερμοκρασία του Ιανουαρίου.

Μια πρώτη προσέγγιση, με τα όσα γνωρίζουμε ως τώρα, θα ήταν να δηλώσουμε 31 μεταβλητές τύπου `float`. Η δήλωση 31 μεταβλητών με το δικό της όνομα η κάθε μια, αφενός μεν είναι πολύ κουραστική, αφετέρου δε δημιουργεί δυσκολία στη διαχείρισή τους. Ο συναθροιστικός τύπος του πίνακα δίνει λύση στο πρόβλημά μας.

4.1.1 Δήλωση Πίνακα

Χρησιμοποιώντας τον τύπο του πίνακα η δήλωσή μας διαμορφώνεται όπως στο σχήμα 4.1.



Σχήμα 4.1

Δήλωση πίνακα 31 στοιχείων τύπου float.

Η δήλωση, η οποία διαβάζεται «ο temp είναι ένας πίνακας με 31 στοιχεία τύπου float», προσδιορίζει το όνομα του πίνακα (`temp`), τον αριθμό των στοιχείων του (31), καθώς και τον τύπο του κάθε στοιχείου (`float`).

4.1.2 Αναφορά στοιχείου Πίνακα

Η αναφορά σε στοιχείο του πίνακα γίνεται με ένα συνδυασμό του ονόματος του πίνακα και ενός αριθμού που προσδιορίζει την τάξη/σειρά του στοιχεί-

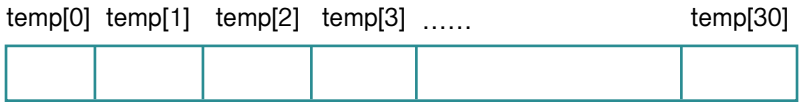
ου μέσα στον πίνακα. Έτσι, για τον πίνακα temp:

το `temp[0]` αναφέρεται στο **πρώτο** στοιχείο του πίνακα,

το `temp[1]` στο **δεύτερο**

και ούτω καθ' εξής μέχρι

το `temp[30]` που αναφέρεται στο **τελευταίο** (Προσοχή! τριακοστό πρώτο) στοιχείο.



ΑΠΟΔΟΣΗ ΑΡΧΙΚΗΣ ΤΙΜΗΣ

Η απόδοση τιμής κατά τη δήλωση στα στοιχεία του πίνακα γίνεται με τη χρήση του τελεστή ανάθεσης. Ο τελεστής ανάθεσης μπαίνει μετά τη διάσταση του πίνακα και ακολουθεί μέσα σε αγκύλες η λίστα με τις τιμές που θα αποδοθούν στα στοιχεία του πίνακα. Η πρόταση

```
float ar[5] = {1 , 2, 3.5, 4, 5};
```

αρχικοποιεί όλα τα στοιχεία του πίνακα `ar` με τις τιμές της λίστας, ενώ η πρόταση

```
float ar[5] = {1 , 2, 3.5 };
```

αρχικοποιεί μόνο τα 3 πρώτα στοιχεία του, δηλαδή τα `ar[0]`, `ar[1]` και `ar[2]`, αφήνοντας τα υπόλοιπα απροσδιόριστα[1].

Σημειώστε τη διαφορετική σημασία του δείκτη στη δήλωση πίνακα από ό,τι στην αναφορά σε στοιχείο πίνακα. Στη δήλωση, ο δείκτης καθορίζει το μέγεθος του πίνακα, ενώ στην αναφορά προσδιορίζει την τάξη μέσα στον πίνακα του στοιχείου στο οποίο αναφερόμαστε.

4.1.3 Το αλφαριθμητικό σαν πίνακας χαρακτήρων

Τα προβλήματα που καλούμαστε να αντιμετωπίσουμε δεν περιέχουν μόνο αριθμητικά δεδομένα. Πολλές είναι οι περιπτώσεις κατά τις οποίες τα δεδομένα ή μέρος των δεδομένων είναι ακολουθίες χαρακτήρων. Για παράδειγμα, ο τίτλος και ο ISBN κωδικός ενός βιβλίου σε ένα σύστημα οργάνωσης

[1] Όπως θα δούμε στη συνέχεια ο μεταγλωττιστής αποδίδει σε αυτά τιμή 0, όταν ο πίνακας έχει δηλωθεί έξω από σώμα συνάρτησης, είναι δηλαδή καθολικός (global).

βιβλιοθήκης εκφράζονται σαν ακολουθίες χαρακτήρων. Τις ακολουθίες αυτές ονομάζουμε *αλφαριθμητικά*. Ο υπολογιστής έχει τη δυνατότητα να επεξεργάζεται, όχι μόνο αριθμητικά, αλλά και αλφαριθμητικά δεδομένα.

Ορισμένες γλώσσες, όπως η Pascal, διαθέτουν ειδικό τύπο για τη διαχείριση των αλφαριθμητικών. Αντίθετα, η C για να αποθηκεύσει και διαχειριστεί αλφαριθμητικά χρησιμοποιεί τον τύπο του πίνακα. Ένα αλφαριθμητικό για την C είναι ένας πίνακας χαρακτήρων που τερματίζει με τον μηδενικό (`null`) χαρακτήρα. Ο μηδενικός χαρακτήρας έχει ASCII κωδικό 0 και αναπαρίσταται με την ακολουθία διαφυγής `'\0'`.

Έτσι, οι δηλώσεις των μεταβλητών για αποθήκευση του τίτλου και του ISBN κωδικού ενός βιβλίου έχουν την παρακάτω μορφή:

```
char book_title[30], isbn[12];
```

4.1.4 Αλφαριθμητική σταθερά

Τα αλφαριθμητικά μπορούν να εμφανίζονται μέσα στον κώδικα όπως οι αριθμητικές σταθερές, αποτελώντας τις αλφαριθμητικές σταθερές. Την αλφαριθμητική σταθερά συναντήσαμε ήδη από το πρώτο μας παράδειγμα για τη γλώσσα C στην υποενότητα 1.2.2, από όπου και θα θυμόσαστε ότι οι χαρακτήρες της περικλείονται σε διπλά εισαγωγικά (όπως απαιτεί η C). Για την αποθήκευσή της, χρησιμοποιείται ένας πίνακας χαρακτήρων με τον μεταγλωττιστή να θέτει αυτόματα στο τέλος του αλφαριθμητικού ένα μηδενικό χαρακτήρα για να προσδιορίσει το τέλος του. Έτσι, η αλφαριθμητική σταθερά `"Hello"` απαιτεί για αποθήκευση 6 bytes όπως φαίνεται παρακάτω:

'H'	'e'	'l'	'l'	'o'	'\0'

Σημειώστε τη διαφορά μεταξύ της σταθεράς χαρακτήρα `'A'` και της αλφαριθμητικής σταθεράς `"A"`. Η πρώτη απαιτεί 1 byte για αποθήκευση, ενώ η δεύτερη δύο, ένα για τον χαρακτήρα A και ένα για τον χαρακτήρα `null`.

ΑΠΟΔΟΣΗ ΑΡΧΙΚΗΣ ΤΙΜΗΣ

Η ανάθεση τιμής με τη δήλωση ακολουθεί τον γενικό κανόνα απόδοσης αρχικής τιμής σε πίνακα. Έτσι, γράφουμε

```
char isbn_str[] = {'0','-','3','8','7','-',
                   '9','7','6','-','1','\0'};
```

Στην πράξη χρησιμοποιείται η εναλλακτική και πιο συμπαγής μορφή με χρήση αλφαριθμητικής σταθεράς:

```
char isbn[] = "0–387–976–1";
```

Προσέξτε ότι στη δήλωση με λίστα στοιχείων, ο προγραμματιστής πρέπει να περιλάβει σαν τελευταία τιμή τον μηδενικό χαρακτήρα. Αντίθετα, στη δήλωση με αλφαριθμητική σταθερά ο μεταγλωττιστής βάζει αυτόματα το '\0'.

Άσκηση Αυτοαξιολόγησης 4.1

Δώστε την πρόταση για εκτύπωση της αλφαριθμητικής σταθεράς `'Hello'` και την πρόταση για εκτύπωση της μεταβλητής *isbn* που ορίσαμε παραπάνω. Θα χρειαστεί να ανατρέξετε στη βασική βιβλιοθήκη της C και, πιο ειδικά, στην τεκμηρίωση της συνάρτησης `printf`.

Άσκηση Αυτοαξιολόγησης 4.2

Δώστε την πρόταση για εισαγωγή τιμής από την κύρια είσοδο στη μεταβλητή *isbn*.

Δραστηριότητα 4.1

Αναπτύξατε ένα πρόγραμμα που θα τυπώνει σε μια γραμμή το όνομα "Δημοσθένης". Στη συνέχεια, να τυπώνει τρεις χαρακτήρες του ονόματος αρχίζοντας από τον δεύτερο και τυπώνοντας έναν χαρακτήρα ανά γραμμή. Τέλος, να τυπώνει τον τελευταίο χαρακτήρα του ονόματος στην τελευταία γραμμή. Μην χρησιμοποιήσετε αλφαριθμητικές σταθερές παρά μόνο για την αρχικοποίηση μιας μεταβλητής. Τη δική μας έκδοση θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 4.2

Τροποποιήστε το πρόγραμμα της Δραστηριότητας 1/Κεφ.4 ώστε να ζητά από τον χρήστη το όνομα του. Για την εμφάνιση του τελευταίου χαρακτήρα του ονόματος χρησιμοποιήστε την συνάρτηση `strlen` της βασικής βιβλιοθήκης. Την δική μας έκδοση θα βρείτε στο τέλος του βιβλίου.

4.1.5 Πολυδιάστατοι πίνακες

Σύμφωνα με τον ορισμό, ένας πίνακας μπορεί να έχει στοιχεία οποιουδήποτε τύπου. Οι παρακάτω προτάσεις δίνουν ορισμένα παραδείγματα:

```
int count[20]; /* η count είναι πίνακας 20 στοιχείων τύπου ακεραίου */
char name[12]; /* η name είναι πίνακας 12 στοιχείων, όπου το καθένα είναι*/
               /* χαρακτήρας */
```

```
int *ar[10];    /* η ar είναι πίνακας 10 στοιχείων, όπου το καθένα είναι*/
                /* δείκτης σε ακέραιο[1] */
```

Αντίστοιχα, ένας πίνακας μπορεί να έχει στοιχεία τα οποία είναι πίνακες. Ένας τέτοιος πίνακας ονομάζεται *πολυδιάστατος*. Η πρόταση

```
int array[4][12];
```

δηλώνει τη μεταβλητή `array` σαν πίνακα 4 στοιχείων, όπου το κάθε ένα από τα τέσσερα στοιχεία της είναι πίνακας 12 στοιχείων τύπου `int`.

Η C δεν βάζει περιορισμό στον αριθμό των διαστάσεων του πίνακα.

Άσκηση Αυτοαξιολόγησης 4.3

Θεωρήστε ότι έχουμε 10 φοιτητές και ο καθένας έχει επιλέξει 4 μαθήματα για τα οποία έχει βαθμολογηθεί από 0 μέχρι 10. Θέλουμε να αναπτύξουμε ένα πρόγραμμα το οποίο θα επεξεργάζεται τους βαθμούς και θα μας δίνει στοιχεία όπως: μέσος όρος κάθε φοιτητή, μικρότερος και μεγαλύτερος βαθμός για κάθε φοιτητή, φοιτητή με τον μεγαλύτερο βαθμό ανά μάθημα κλπ. Δώστε τις δηλώσεις των απαραίτητων μεταβλητών.

Άσκηση Αυτοαξιολόγησης 4.4

Δηλώστε ένα πίνακα για τη διαχείριση των μέσων ημερήσιων θερμοκρασιών των μηνών των τελευταίων 5 ετών. Στη συνέχεια, δώστε την έκφραση που η τιμή της είναι η θερμοκρασία της 1ης Δεκεμβρίου του τελευταίου έτους.

4.2 Ο τύπος του δείκτη

Όπως ήδη γνωρίζετε, κάθε μεταβλητή σχετίζεται με μία θέση στην κύρια μνήμη του υπολογιστή. Κάθε θέση στη μνήμη έχει τη δική της ξεχωριστή διεύθυνση. Με άμεση χρήση αυτής της διεύθυνσης, γίνεται στη γλώσσα μηχανής, η αναφορά στα δεδομένα είτε για αποθήκευση είτε για ανάκλησή τους. Στις γλώσσες υψηλού επιπέδου ο προγραμματιστής, όπως ήδη έχουμε αναφέρει, δε βλέπει διευθύνσεις αλλά αναφέρεται στα δεδομένα με τη χρήση συμβολικών ονομάτων, τα οποία το σύστημα αντιστοιχεί στις πραγματικές διευθύνσεις.

Εντούτοις, ορισμένες γλώσσες υψηλού επιπέδου, θέλοντας να δώσουν στον προγραμματιστή τη δυνατότητα δημιουργίας πολύ αποδοτικού κώδικα, εισή-

[1] Προσέξτε τη μεγαλύτερη προτεραιότητα του τελεστή [ ] από τον *.

γαγαν την έννοια του δείκτη. Ο δείκτης δεν είναι τίποτε παραπάνω παρά μία μεταβλητή η οποία χρησιμοποιείται για την αποθήκευση μιας διεύθυνσης της κύριας μνήμης του υπολογιστή.

4.2.1 Δήλωση δείκτη

Για τη δήλωση μιας μεταβλητής δείκτη πρέπει να γνωστοποιήσουμε στον μεταγλωττιστή ότι η μεταβλητή είναι τύπου δείκτη, καθώς και τον τύπο του αντικειμένου που μπορεί να δείχνει η μεταβλητή. Η δήλωση μιας μεταβλητής δείκτη για τη γλώσσα C έχει την παρακάτω μορφή:

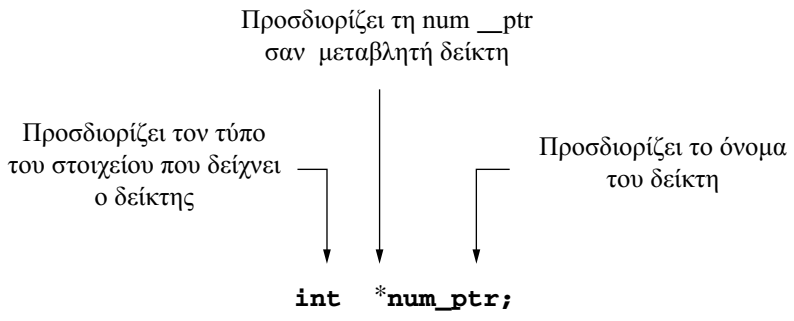
<όνομα τύπου> *<όνομα δείκτη>;

Στο σχήμα 4.2 δίνουμε τη δήλωση μιας μεταβλητής δείκτη ή ενός δείκτη, όπως θα λέμε πιο απλά, στη συνέχεια. Ο δείκτης αυτός έχει όνομα `num_ptr` και δείχνει σε αντικείμενο ακέραιου τύπου. Το σύμβολο * είναι ο **τελεστής περιεχομένου** και μπορεί να ερμηνευθεί στη δήλωση `int *num_ptr;` κατά δύο τρόπους:

α) η μεταβλητή `num_ptr` είναι δείκτης σε `int` ή

β) το περιεχόμενο της θέσης μνήμης που δείχνει ο num_ptr είναι int.

Παρατηρήστε τη χρήση του αναγνωριστικού `_ptr` στο τέλος του ονόματος της μεταβλητής δείκτη. Βελτιώνει την αναγνωσιμότητα του κώδικα. Ο προγραμματιστής δεν χρειάζεται να ανατρέξει στη δήλωση της μεταβλητής για να δει αν αυτή είναι δείκτης ή όχι.



Σχήμα 4.2
Δήλωση δείκτη.

4.2.2 Αρχικοποίηση δείκτη

Μετά τη δήλωσή του, ένας δείκτης πρέπει να δείχνει σε μια θέση μνήμης που ανήκει στο πρόγραμμα. Για το λόγο αυτό δίνουμε συνήθως τιμή στο δείκτη

ταυτόχρονα με τη δήλωσή του. Στην περίπτωση αυτή, η δήλωση δείκτη παίρνει τη μορφή:

<όνομα τύπου> *<όνομα δείκτη> = <διεύθυνση>;

όπου <διεύθυνση> μπορεί να είναι:

α) μία άμεση διεύθυνση[1] όπως στην πρόταση

```
int *num_ptr = 1000;  /* ο δείκτης num_ptr δείχνει ακέραιο στη θέση */
                      /* μνήμης 1000 */
```

β) η διεύθυνση μιας μεταβλητής ακέραιου τύπου την οποία παίρνουμε εφαρμόζοντας μπροστά από μια μεταβλητή τον **τελεστή διεύθυνσης** `&`, όπως στην πρόταση

```
int *num_ptr = &num;  /* ο δείκτης num_ptr δείχνει στη μεταβλητή num */
```

4.2.3 Ανάθεση τιμής σε δείκτη

Αν ένας δείκτης δεν αρχικοποιηθεί, θα πρέπει σύντομα και οπωσδήποτε πριν από τη χρήση του, να του ανατεθεί τιμή με μια πρόταση ανάθεσης της μορφής

<όνομα δείκτη> = <διεύθυνση>;

για παράδειγμα, `num_ptr = 1000;` ή `num_ptr = &num;`

Αποφύγετε αρχικοποιήσεις με άμεση διεύθυνση. Είναι πολύ επικίνδυνες και χρησιμοποιούνται, συνήθως, μόνο για άμεση πρόσβαση στο υλικό.

4.2.4 Τελεστές που έχουν σχέση με δείκτες

Θα αναρωτηθήκατε ίσως «Μα ο τελεστής * δεν αναπαριστά τον πολλαπλασιασμό;». Δεν έχετε άδικο. Προσέξτε λοιπόν, τι είναι αυτό που του δίνει διαφορετική σημασία. Στην περίπτωση του πολλαπλασιασμού, το σύμβολο χρειάζεται δύο τελεστέους, τις τιμές των οποίων πολλαπλασιάζει μεταξύ τους, ενώ στην περίπτωση που εφαρμόζεται πάνω σε μία μεταβλητή δείκτη δίνει το περιεχόμενο της θέσης μνήμης που αυτός δείχνει.

Είναι χρήσιμο στο σημείο αυτό να δούμε συγκεντρωμένους τους τελεστές που μέχρι τώρα συναντήσαμε και έχουν σχέση με δείκτες.

---

[1] Η αναπαράσταση της διεύθυνσης εξαρτάται από τον υπολογιστή με πλέον συνηθισμένη μονάδα αποθήκευσης το byte.

Τελεστής	Σημασία
=	ο τελεστής ανάθεσης χρησιμοποιείται για ανάθεση τιμής σε δείκτη
*	ο τελεστής περιεχομένου **εφαρμόζεται μόνο σε δείκτη** και έχει σαν αποτέλεσμα το περιεχόμενο της διεύθυνσης που δείχνει ο δείκτης
&	ο τελεστής διεύθυνσης δίνει τη διεύθυνση της μεταβλητής στην οποία εφαρμόζεται

Άσκηση Αυτοαξιολόγησης 4.5

Με δεδομένες τις δηλώσεις `int year=1821;` και `int *year_ptr=&year;` συμπληρώστε τις παρακάτω αντιστοιχήσεις

`year`	διεύθυνση της μεταβλητής `year_ptr`
`&year`	
`*year`	1821
`*year_ptr`	
`year_ptr`	διεύθυνση της μεταβλητής `year`
`&year_ptr`	

Άσκηση Αυτοαξιολόγησης 4.6

Με δεδομένες τις προτάσεις `int num1; int *num2; int * pt1, pt2;` αντιστοιχήστε κάθε μεταβλητή με τον τύπο της.

`num1`	ακέραια μεταβλητή
`num2`	
`pt1`	μεταβλητή δείκτη
`pt2`	

Άσκηση Αυτοαξιολόγησης 4.7

Μετά τη δήλωση της ακέραιας μεταβλητής `num` με την πρόταση `int num;` και τη δήλωση του δείκτη `num_ptr` με τη δήλωση `int *num_ptr;` ο προγραμματιστής θέλει να βάλει το δείκτη `num_ptr` να δείχνει στην ακέραια μεταβλητή `num`. Ποια από τις παρακάτω προτάσεις είναι η κατάλληλη.

Α. `num_ptr = num;`
Β. `*num_ptr = &num;`
Γ. `num_ptr = &num;`
Δ. `*num_ptr = num;`

4.2.5 Σχέση δεικτών με πίνακες

Όπως ήδη αναφέραμε, οποιαδήποτε λειτουργία στη C γίνεται με πίνακες, μπορεί να γίνει και με δείκτες. Έτσι, αν `arr` είναι πίνακας δέκα ακεραίων `int arr[10];` και ο `p` δείκτης σε ακέραιο `int *p;` μπορούμε να αναθέσουμε στον `p` να δείχνει στο πρώτο στοιχείο του arr με την παρακάτω πρόταση

```
p = &arr[0];
```

ή εναλλακτικά και εκμεταλλευόμενοι το γεγονός ότι το όνομα του πίνακα είναι η διεύθυνση του πρώτου στοιχείου του, μπορούμε να γράψουμε

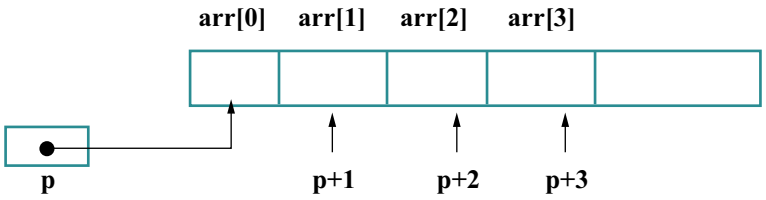
```
p = arr;
```

Αν εφαρμόσω τον τελεστή `++` στο δείκτη `p` αυτός θα δείχνει, μετά την εφαρμογή, στο επόμενο στοιχείο του πίνακα. Προσέξτε ότι δεν μπορώ αντίστοιχα με το `p++` να γράψω `arr++` γιατί το `arr` είναι σταθερά διεύθυνση. Όπως φαίνεται και στο σχήμα 4.3 ισχύουν τα παρακάτω:

η έκφραση `arr + 1` δείχνει το στοιχείο τάξης 1 του πίνακα `arr`,

η έκφραση `arr + 2` δείχνει το στοιχείο τάξης 2 του πίνακα `arr`, και γενικά

η έκφραση `arr + n` δείχνει το στοιχείο τάξης n του πίνακα `arr`.



Σχήμα 4.3.
Χρήση δεικτών για πρόσβαση στοιχείων πίνακα.

Εκμεταλλευόμενοι το γεγονός αυτό μπορούμε να αναφερθούμε στο στοιχείο `arr[n]` χρησιμοποιώντας την έκφραση `*(arr + n)`. Ισχύουν δηλαδή οι ισοδυναμίες:

```
*(arr + n) <—> arr[n]
 arr + n <—> &arr[n]
```

Σύνοψη

Αναφερθήκαμε στο συναθροιστικό τύπο του πίνακα και είδαμε πώς αυτός χρησιμοποιείται για να ομαδοποιήσει δεδομένα κοινού τύπου. Ο πίνακας αποτελεί ένα από τα πολύ χρήσιμα στοιχεία της C, υποστηρίζοντας εκτός των άλλων και τη διαχείριση των αλφαριθμητικών. Ιδιαίτερη προσοχή θα πρέπει να δοθεί στην αναφορά του πρώτου στοιχείου του πίνακα που θεωρείται σαν τάξης 0 και όχι 1, καθώς και στο τελευταίο που θεωρείται σαν τάξης n–1 με n το μέγεθος του πίνακα. Είδαμε, τέλος, πως ο δείκτης μας παρέχει έναν εναλλακτικό τρόπο αναφοράς στην πληροφορία που είναι αποθηκευμένη στην κύρια μνήμη του υπολογιστή.

Βιβλιογραφία

[King ’96]

King K.N., «*C Programming: A modern approach*», W.W.Norton & Company, Inc.

ΚΕΦΑΛΑΙΟ 5

Τελεστές – Εκφράσεις – Προτάσεις

Σκοπός

Σκοπός του κεφαλαίου είναι να εισάγει τη βασική έννοια του τελεστή και του τρόπου που αυτός χρησιμοποιείται αφενός μεν για το σχηματισμό εκφράσεων, αφετέρου δε για τον υπολογισμό της τιμής αυτών των εκφράσεων. Εισάγεται επιπλέον η πρόταση σα βασική μονάδα δόμησης του προστακτικού προγραμματισμού.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *αναφέρετε 4 σημειολογίες που χρησιμοποιούνται για τη δημιουργία εκφράσεων,*
- *δημιουργείτε εκφράσεις και να υπολογίζετε την τιμή τους,*
- *δημιουργείτε το δένδρο αφηρημένης σύνταξης δεδομένης έκφρασης,*
- *περιγράψετε τον υπολογισμό περιορισμένης έκτασης και να δώσετε χαρακτηριστικά παραδείγματα,*
- *δώσετε 3 τουλάχιστον παραδείγματα αυτόματης υπονοούμενης μετατροπής τύπων,*
- *δώσετε 2 τουλάχιστον παραδείγματα ρητής μετατροπής τύπων,*
- *εξηγήσετε τη σημασία των επιπέδων προτεραιότητας των τελεστών,*
- *εξηγήσετε με παραδείγματα τη σημασία της προσεταιριστικότητας των τελεστών,*
- *δώσετε από 2 τουλάχιστον εκφράσεις για κάθε έναν από τους τελεστές της C.*

Έννοιες–Κλειδιά

- *σημειολογία ένθετου τελεστή (infix)*
- *σημειολογία προπορευόμενου τελεστή (prefix)*
- *σημειολογία παρελκόμενου τελεστή (postfix)*
- *λογικός τελεστής*
- *υπονοούμενες μετατροπές (implicit conversions)*
- *ρητές μετατροπές (explicit conversions)*
- *συσχετιστικός τελεστής*
- *εφαρμοστική σειρά (applicative order)*

- *υπολογισμός περιορισμένης κλίμακας(short circuit evaluation)*
- *ελεγχόμενος πλεονασμός*
- *προτεραιότητα (precedence)*
- *προσεταιριστικότητα (associativity)*
- *αριθμητικός τελεστής*
- *δένδρο αφηρημένης σύνταξης (abstract syntax tree)*
- *μετατροπή τύπων*
- *παρενέργειες (side effects)*
- *πρόταση (statement)*
- *σύνθετη πρόταση*
- *εντολή (command)*

Εισαγωγικές Παρατηρήσεις

*Σίγουρα έχετε χρησιμοποιήσει εκφράσεις της μορφής (100+45)*4 ή ακόμη πιο σύνθετες, όπως η (x + y) * z –10. Εκφράσεις αυτής της μορφής χρησιμοποιούνται από τον άνθρωπο για αιώνες τώρα και αποτέλεσαν σημείο αναφοράς για το σχεδιασμό των γλωσσών προγραμματισμού. Από τα μαθηματικά, εξάλλου, γνωρίζετε ότι το βασικό στοιχείο για το σχηματισμό των εκφράσεων είναι ο τελεστής. Ο τελεστής αναπαριστά μια διεργασία που εκτελείται πάνω σε ένα ή περισσότερα δεδομένα, τα οποία καλούνται τελεστέοι και μπορεί να έχουν τη μορφή μεταβλητών ή σταθερών. Το αποτέλεσμα της διεργασίας είναι ο υπολογισμός μιας τιμής, η οποία και αποτελεί την τιμή της έκφρασης.*

*Για τον υπολογισμό εκφράσεων που περιλαμβάνουν περισσότερους από έναν τελεστή, όπως η έκφραση x + y * z, θα πρέπει να κάνετε χρήση του κανόνα που μάθατε στα μαθηματικά, και ο οποίος ορίζει ότι ο τελεστής * έχει μεγαλύτερη προτεραιότητα από τον τελεστή +. Αυτό σημαίνει ότι θα εκτελεστεί πρώτα η διεργασία του πολλαπλασιασμού πάνω στους τελεστέους y και z και, στη συνέχεια, η διεργασία της πρόσθεσης πάνω στο αποτέλεσμα του πολλαπλασιασμού και το x. Είναι, επομένως, απαραίτητη η εισαγωγή της έννοιας της προτεραιότητας. Η προτεραιότητα ορίζει την σειρά με την οποία εφαρμόζονται οι τελεστές ή πιο απλά τη σειρά με την οποία εκτελούνται οι διεργασίες που αναπαριστούν οι τελεστές. Την προτεραιότητα αυτή, όπως σίγουρα γνωρίζετε, μπορώ να μεταβάλω χρησιμοποιώντας παρενθέσεις. Έτσι, στην έκφραση (x + y) * z οι παρενθέσεις επιβάλουν την εφαρμογή του τελεστή + και, στη συνέχεια, την εφαρμογή του τελεστή *.*

Όλα αυτά βρίσκουν εφαρμογή στις γλώσσες προγραμματισμού, με αποτέλεσμα να είστε πολύ σύντομα σε θέση να γράψετε τις πρώτες σας εκφράσεις που θα αποτελέσουν τις προτάσεις των πρώτων σας προγραμμάτων.

Το κεφάλαιο αυτό αποτελείται από πέντε ενότητες. Στην πρώτη από αυτές παρουσιάζεται ο τελεστής σαν δομικό στοιχείο της γλώσσας το οποίο χρησιμοποιείται για να αναπαραστήσει τις βασικές διεργασίες που η γλώσσα άμεσα υποστηρίζει. Η δεύτερη ενότητα αναφέρεται στους συμβολισμούς που οι γλώσσες χρησιμοποιούν για το σχηματισμό εκφράσεων, καθώς και τον τρόπο υπολογισμού της τιμής αυτών των εκφράσεων, εισάγοντας τους όρους της προτεραιότητας και προσεταιριστικότητας. Η τρίτη ενότητα αναφέρεται στο πολύ σημαντικό θέμα της μετατροπής τύπων και τον τρόπο που ο μηχανισμός μετατροπής τύπων διευκολύνει τη δημιουργία εκφράσεων από διαφορετικού τύπου τελεστέους. Η τέταρτη ενότητα εισάγει την έννοια της πρότασης κάνοντας μια σύντομη αναφορά στις προτάσεις του προεπεξεργαστή της C. Τέλος, η πέμπτη ενότητα αναφέρεται στο πολύ πλούσιο σύνολο τελεστών που διαθέτει η C.

5.1 Ο τελεστής στη γλώσσα προγραμματισμού

Ένας *τελεστής* (operator) είναι ένα σύμβολο ή μία λέξη της γλώσσας προγραμματισμού, που αναπαριστά συγκεκριμένη διεργασία, η οποία εκτελείται πάνω σε ένα ή περισσότερα δεδομένα. Τα δεδομένα καλούνται *τελεστέοι* (operands) και μπορούν να είναι μεταβλητές, σταθερές ή ακόμη κλήσεις συναρτήσεων. Οι τελεστές χρησιμοποιούνται για το σχηματισμό εκφράσεων (expressions). Στην έκφραση `num + 12` ο χαρακτήρας `+` αναπαριστά τη διεργασία της πρόσθεσης των δύο τελεστέων, της τιμής της μεταβλητής `num` και της σταθεράς `12`.

Ο προγραμματιστής χρησιμοποιεί τους τελεστές για να επεξεργαστεί δεδομένα και να πάρει τα αποτελέσματα της επεξεργασίας. Αυτό το πετυχαίνει κατασκευάζοντας εκφράσεις. Για να εκτελέσει, για παράδειγμα, την επεξεργασία της σύγκρισης των μεταβλητών `num1` και `num2` κατασκευάζει την έκφραση `num1 > num2` κάνοντας χρήση του συσχετιστικού τελεστή `>`, ενώ για την αναφορά στο πέμπτο στοιχείο του πίνακα `ar` δημιουργεί, για την C, την έκφραση `ar[4]` κάνοντας χρήση του τελεστή `[]`.

Οι τελεστές ταξινομούνται, ανάλογα με τον αριθμό των τελεστέων στους οποίους δρουν, σε μοναδιαίους (unary), δυαδικούς (binary) και τριαδικούς (ternary). Ταξινομούνται, επίσης, και ανάλογα με τη διεργασία που εκτελούν σε κατηγορίες οι σημαντικότερες των οποίων δίνονται στον πίνακα 5.1. Ο πίνακας δίνει παράλληλα και ενδεικτικούς τελεστές κάθε κατηγορίας για τη C, η οποία έχει το προνόμιο να διαθέτει ένα πολύ πλούσιο σύνολο τελεστών πλουσιότερο από αυτό της Pascal.

Πίνακας 5.1

Κατηγορίες τελεστών.

Κατηγορία	ενδεικτικοί τελεστές
αριθμητικοί	+ – * / %
λογικοί	&& || !
συσχετιστικοί	> >= == !=
διαχείρισης των bits ενός byte	>> & | ^ ~
τελεστές διαχείρισης μνήμης	& [] . –>

Το σύνολο των τελεστών προσδιορίζει και τις βασικές διεργασίες που η γλώσσα προγραμματισμού άμεσα υποστηρίζει. Οι διεργασίες αυτές, βέβαια, είναι στοιχειώδεις συγκρινόμενες με τις διεργασίες που τα πραγματικά συστήματα εκτελούν.

5.2 Έκφραση

Μία έκφραση, στη γενική της περίπτωση, αποτελείται από έναν ή περισσότερους τελεστέους και, προαιρετικά, από ένα ή περισσότερους τελεστές. Οι τελεστέοι, μεταβλητές, σταθερές, και κλήσεις συναρτήσεων αποτελούν από μόνοι τους εκφράσεις, αλλά μπορούν να συνδυαστούν με τους τελεστές για να σχηματίσουν σύνθετες εκφράσεις. Ο υπολογισμός της τιμής κάθε έκφρασης γίνεται σύμφωνα με κανόνες, ορισμένοι από τους οποίους είναι απλοί και ευνόητοι, όπως οι κανόνες που εφαρμόζονται για τον υπολογισμό των παρακάτω απλών και σύνθετων εκφράσεων, και άλλοι είναι περισσότερο σύνθετοι και θέλουν ιδιαίτερη αναφορά.

Απλές εκφράσεις	σύνθετες εκφράσεις
`8`	`12 * 20`
`count`	`count + 1`
`func()`	`num / (count + 1)`
`MAX_VALUE`	`func() / 4`

5.2.1 Συμβολισμοί στο σχηματισμό εκφράσεων

Οι γλώσσες προγραμματισμού χρησιμοποιούν διάφορους συμβολισμούς για το σχηματισμό εκφράσεων με πιο συνηθισμένο το συμβολισμό που τοποθετεί τον τελεστή μεταξύ των τελεστέων. Ειδικότερα, ένας δυαδικός τελεστής μπορεί να τοποθετηθεί

1. μεταξύ των δεδομένων στα οποία ενεργεί, όπως στην έκφραση `x + y`, οπότε έχουμε τη σημειολογία **ένθεσης** ή ένθετου τελεστή (infix notation),
2. πριν από τους τελεστέους όπως στην έκφραση `+x y`, οπότε έχουμε τη σημειολογία **πρόθεσης** ή προπορευόμενου τελεστή (prefix notation) και
3. μετά από τους τελεστέους όπως στην έκφραση `x y +`, οπότε έχουμε τη σημειολογία **επίθεσης** ή παρελκόμενου τελεστή (postfix notation)[1].

Σε κάθε περίπτωση όμως, η τιμή που προκύπτει από την εφαρμογή της διεργασίας που αναπαριστά ο τελεστής είναι η ίδια. Η τιμή αυτή δεν αλλάζει και στην περίπτωση που η έκφραση κλεισθεί σε παρενθέσεις. Είναι προφανές ότι η έκφραση `x + y` έχει την ίδια τιμή με την έκφραση[2] `(x + y)`.

Η σημειολογία του ένθετου τελεστή παράγει κατανοητές και εύκολες στην ανάγνωση εκφράσεις, ενώ αυτή του παρελκόμενου τελεστή έχει το πλεονέκτημα του ευκολότερου μηχανικού υπολογισμού με τη χρήση της έννοιας της στοίβας, κάτι που θα δούμε στην υποενότητα 8.1.3.

5.2.2 Κατηγορίες εκφράσεων της C

Τις εκφράσεις της C μπορούμε να κατατάξουμε σύμφωνα με τον Darnel [Darnel 91] στις παρακάτω κατηγορίες:

- **Σταθερές εκφράσεις**. Είναι εκφράσεις που περιέχουν μόνο σταθερές τιμές.
- **Ακέραιες εκφράσεις** και **εκφράσεις κινητής υποδιαστολής**. Είναι εκφράσεις οι οποίες μετά από όλες τις άμεσες και έμμεσες μετατροπές τύπων δίνουν αποτέλεσμα ακέραιου τύπου ή τύπου κινητής υποδιαστολής αντίστοιχα.
- **Εκφράσεις δείκτη**. Είναι εκφράσεις των οποίων η τιμή είναι διεύθυνση. Οι εκφράσεις αυτές περιλαμβάνουν μεταβλητές δείκτη, τον τελεστή διεύθυνσης `&`, αλφαριθμητικές σταθερές και ονόματα πινάκων. Έτσι, αν `mum_ptr` είναι δείκτης και η `num` ακέραια μεταβλητή, οι παρακάτω εκφράσεις είναι εκφράσεις δείκτη

[1] Οι σημειολογίες προπορευόμενου και παρελκόμενου τελεστή ονομάζονται και ελεύθερες παρενθέσεων (parenthesis–free) σημειολογίες, γιατί οι τελεστέοι κάθε τελεστή προσδιορίζονται σαφώς χωρίς τη χρήση παρενθέσεων.

[2] Αυτό δεν είναι αληθές για την συναρτησιακή γλώσσα Lisp, η οποία αποτελεί εξαίρεση στον κανόνα αυτό.

num_ptr	num_ptr + 1	0xFF00[1]
&num	"Hello World"	&num + 2

5.2.3 Υπολογισμός τιμής έκφρασης

Για τον υπολογισμό της τιμής μιας έκφρασης, ουσιαστικής σημασίας είναι οι έννοιες της **προτεραιότητας** (precedence) και της **προσεταιριστικότητας** (associativity), οι οποίες επηρεάζουν τον τρόπο με τον οποίο οι τελεστέοι προσδένονται στους τελεστές.

Κάθε γλώσσα, ταξινομεί τους τελεστές της σε επίπεδα προτεραιότητας, με τη σύμβαση ότι οι τελεστές μεγαλυτέρου επιπέδου προτεραιότητας ενεργούν στους τελεστέους πριν από τους αντίστοιχους χαμηλότερου επιπέδου. Έτσι, οι περισσότερες ευρέως χρησιμοποιούμενες γλώσσες έχουν τους τελεστές * και / στο ίδιο επίπεδο προτεραιότητας, και τους + και – σε άλλο χαμηλότερο[2]. Η C διαθέτει, σε σύγκριση με την Pascal, όχι μόνο πλουσιότερο σύνολο τελεστών αλλά και επιπέδων προτεραιότητας.

Η ύπαρξη περισσότερων τελεστών στο ίδιο επίπεδο, επιβάλει τον προσδιορισμό της κατεύθυνσης εφαρμογής, με την από αριστερά προς τα δεξιά κατεύθυνση να είναι ευρύτερα χρησιμοποιούμενη. Ένας τελεστής, λέμε πως είναι *αριστερής προσεταιριστικότητας* (left associative), όταν σε εκφράσεις που περιέχουν πολλά στιγμιότυπα του τελεστή, η εφαρμογή γίνεται από αριστερά προς τα δεξιά. Έτσι, η έκφραση `10 – 8 –2` υπολογίζεται σαν `(10 – 8) – 2`. Οι τελεστές +, –, * και / είναι όλοι αριστερής προσεταιριστικότητας.

Παράδειγμα τελεστή δεξιάς προσεταιριστικότητας αποτελεί, για την C, η ύψωση σε δύναμη καθώς και ο τελεστής ανάθεσης (=). Έτσι, στην έκφραση

```
num1 = num2 = 10
```

εφαρμόζεται πρώτα ο δεξιός τελεστής ανάθεσης, με αποτέλεσμα να ανατεθεί η τιμή 10 στη μεταβλητή `num2`. Στη συνέχεια, εφαρμόζεται ο αριστερός τελεστής που έχει σαν αποτέλεσμα την ανάθεση της τιμής του `num2` στη μεταβλητή `num1`. Τελικά, η έκφραση που είναι πολύ συνηθισμένη στην C έχει σαν αποτέλεσμα την αρχικοποίηση των δύο μεταβλητών με την τιμή 10. Ο πίνακας 5.2 δίνει τον πλήρη πίνακα προτεραιοτήτων και προσεταιριστικότητας των τελεστών της C.

---

[1] Αποτελεί άμεση διεύθυνση του υλικού.

[2] Η Smalltalk–80 είναι μια από τις λίγες εξαιρέσεις του κανόνα αυτού. Έχει όλους τους αριθμητικούς τελεστές στο ίδιο επίπεδο με αποτέλεσμα η έκφραση x + y * z να υπολογίζεται σαν (x + y) * z.

Πίνακας 5.2.

Προτεραιότητα και προσεταιριστικότητα τελεστών της C.

Τελεστές	Προσεταιριστικότητα
( ) [ ] –> .	από αριστερά προς τα δεξιά
! ~ ++ — + – * & (τύπος)	από δεξιά προς τα αριστερά
sizeof	από αριστερά προς τα δεξιά
* / %	από αριστερά προς τα δεξιά
+ –	από αριστερά προς τα δεξιά
<< >>	από αριστερά προς τα δεξιά
< <= > >=	από αριστερά προς τα δεξιά
== !=	από αριστερά προς τα δεξιά
&	από αριστερά προς τα δεξιά
^	από αριστερά προς τα δεξιά
|	από αριστερά προς τα δεξιά
&&	από αριστερά προς τα δεξιά
||	από αριστερά προς τα δεξιά
?:	από δεξιά προς τα αριστερά
= += –= *= /= %= &= ^= |= <<= >>=	από δεξιά προς τα αριστερά

Οι παραπάνω κανόνες προτεραιότητας και προσεταιριστικότητας τηρούνται με ελάχιστες εξαιρέσεις από πολλές γλώσσες μεταξύ των οποίων οι Pascal, Ada, Fortran, PL/1, ALGOL και Java. Στις γλώσσες αυτές δεν είναι απαραίτητη η χρήση παρενθέσεων για τον προσδιορισμό του τρόπου υπολογισμού της τιμής των εκφράσεων. Παρόλα αυτά, χρησιμοποιούμε παρενθέσεις

- είτε για να προσδιορίσουμε συγκεκριμένη σειρά εφαρμογής, όπως στην έκφραση `(2 –3) * 4`,
- είτε για να αυξήσουμε την αναγνωσιμότητα μιας έκφρασης όπως στην `2 – (3 * 4)`, παρά το γεγονός ότι στην περίπτωση αυτή αποτελεί πλεονασμό.

Στην περίπτωση ένθετων παρενθέσεων, ο μεταγλωττιστής εφαρμόζει πρώτα τις εσωτερικές παρενθέσεις. Η σειρά υπολογισμού όμως δεν ορίζεται για παρενθέσεις που είναι στο ίδιο βάθος ένθεσης. Επιπλέον, για εκφράσεις όπως η `x + y + z * w` δεν καθορίζεται αν θα εκτελεστεί πρώτα η x + y και στη συνέχεια η `z * w` ή αντίστροφα, ή ακόμη, με τη σειρά που δείχνουν στη συνέχεια οι παρενθέσεις `(x + (y + (z * w)))` όπου πρώτα υπολογίζεται το `z * w`, το

αποτέλεσμα αθροίζεται με το `y` και το νέο αποτέλεσμα αθροίζεται με το `x`.

Ο υπολογισμός μιας έκφρασης της μορφής *τελ1* ***τελεστής*** *τελ2* έχει γενικά σαν αποτέλεσμα τον υπολογισμό των δύο τελεστέων τελ1 και τελ2 και στη συνέχεια την εφαρμογή του τελεστή στα αποτελέσματα. Αυτή η σειρά υπολογισμού ονομάζεται **εφαρμοστική σειρά** (applicative order). Σε ορισμένες όμως περιπτώσεις, και ειδικά στους λογικούς τελεστές, είναι αποτελεσματικότερη η εφαρμογή μιας διαφορετικής τεχνικής υπολογισμού, γνωστής με το όνομα **υπολογισμός περιορισμένης έκτασης** (short circuit evaluation). Η τεχνική αυτή έχει σαν αποτέλεσμα τον μη υπολογισμό του *y*

α) στην έκφραση ***x and y***, όταν το *x* είναι ψευδές και

β) στην έκφραση ***x or y*** όταν το *x* είναι αληθές.

Η C υλοποιεί τον υπολογισμό περιορισμένης έκτασης για τους λογικούς τελεστές `AND` και `OR`, αντίθετα η Ada, όπως και ορισμένες άλλες γλώσσες, έχουν ξεχωριστούς λογικούς τελεστές για περιπτώσεις που ο υπολογισμός περιορισμένης έκτασης είναι επιθυμητός.

Άσκηση Αυτοαξιολόγησης 5.1

Σε ποια από τις κατηγορίες εκφράσεων της C ανήκει η έκφραση που αποτελείται από το όνομα ενός πίνακα ακεραίων;

Άσκηση Αυτοαξιολόγησης 5.2

Εφαρμόστε τους κανόνες προτεραιότητας και προσεταιριστικότητας των τελεστών για να υπολογίσετε την τιμή της έκφρασης `2 + ((4 +2)/(7–5) – 6)`.

Άσκηση Αυτοαξιολόγησης 5.3

Να γίνει η αντιστοίχηση των εκφράσεων της δεξιάς στήλης με τις τιμές της αριστερής.

Έκφραση	**τιμή**
`–4 + 6`	true (1)
`c=3+8`	17
`x`	2
`sqrt(16.0)`	η τιμή της x
`1821`	false
`5>3`	4.0
`8<1`	11
`6 + (c=3+8)`	false (0)
`(5> 1) && (6>7)`	1821

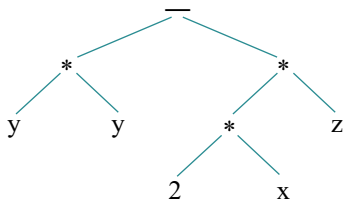
5.2.4 Δένδρο αφηρημένης σύνταξης

Μια από τις μεθόδους που οι μεταγλωττιστές χρησιμοποιούν για τον υπολογισμό εκφράσεων είναι η δημιουργία μιας δομής ανάποδου δένδρου, όπως αυτό που δίνεται στο σχήμα 5.1. Κάθε τελεστής αποτελεί έναν *κόμβο* (node) του δένδρου και δείχνει στους τελεστέους του, οι οποίοι καλούνται *φύλλα* (leaves). Ο μεταγλωττιστής υπολογίζει την έκφραση ξεκινώντας από το κάτω μέρος του δένδρου. Για κάθε τελεστή που εφαρμόζει, τοποθετεί το αποτέλεσμα που βρήκε στη θέση του κόμβου ώστε να αποτελέσει τελεστέο για τον τελεστή του αμέσως ανώτερου επιπέδου.

Τα δένδρα που δείχνουν τη δομή μιας έκφρασης όσον αφορά τους τελεστές και τελεστέους, ονομάζονται δένδρα αφηρημένης σύνταξης (abstract syntax trees), γιατί δείχνουν τη συντακτική δομή μιας έκφρασης ανεξάρτητα από τη σημειολογία που χρησιμοποιείται για τη σύνταξη της έκφρασης.

Σχήμα 5.1

Δένδρο αφηρημένης σύνταξης της έκφρασης `y * y – 2 * x * z`



Έτσι το δένδρο του σχήματος 5.1 αποτελεί το δένδρο αφηρημένης σύνταξης της διεργασίας που μπορεί να αναπαρασταθεί από τις παρακάτω εκφράσεις στις διάφορες σημειολογίες

Prefix notation : `– * y y * * 2 x z`

Infix notation: `y * y – 2 * x * z`

Postfix notation: `y y * 2 x * z * –`

Μια ενδιαφέρουσα παραλλαγή του προθεματικού τελεστή είναι η έκφραση `max(a,b)`, όπου ο τελεστής `max` που αναπαριστά τη διεργασία της εύρεσης του μεγαλύτερου, γράφεται στα αριστερά και οι τελεστέοι πάνω στους οποίους ενεργεί ακολουθούν σε παρένθεση και διαχωρίζονται μεταξύ τους με κόμμα. Ο συμβολισμός αυτός, που επιτρέπει σε τελεστές να δέχονται μεταβλητό αριθμό τελεστέων, αποτελεί, όπως θα δούμε στη συνέχεια, τη βάση πάνω στην οποία δομείται ο μηχανισμός των συναρτήσεων της C.

Άσκηση Αυτοαξιολόγησης 5.4

Υπολογίστε την τιμή της έκφρασης `2 + ((4 +2)/(7–5) – 6)` χρησιμοποιώντας την τεχνική του δένδρου αφηρημένης σύνταξης.

5.3 Μετατροπές τύπων

Μια ερώτηση που πιθανόν σας απασχολεί είναι η ακόλουθη: «Μπορεί ένας τελεστής να εφαρμοστεί σε τελεστέους διαφορετικού τύπου;» ή διαφορετικά «Μπορώ να προσθέτω πορτοκάλια με μήλα;». Στην ερώτηση της δασκάλας «Πόσο κάνουν 2 πορτοκάλια και 3 μήλα» τα παιδιά θα απαντούσαν «δεν μπορώ να προσθέσω διαφορετικά πράγματα μεταξύ τους», ενώ ο Τοτός θα έλεγε «κυρία μας κάνουν 5 φρούτα». Ας δούμε ποια από τις δύο απαντήσεις είναι η σωστή. Οι περισσότεροι δυαδικοί τελεστές απαιτούν από τους τελεστέους τους να είναι του ίδιου τύπου και, επομένως, τα παιδιά απάντησαν σωστά. Παρόλα αυτά, η C επιτρέπει με ελάχιστους περιορισμούς την ανάμειξη των αριθμητικών τύπων σε εκφράσεις, και αυτό γιατί ο μεταγλωττιστής αναλαμβάνει να μετατρέψει αυτόματα τους τελεστέους ώστε να είναι ίδιου τύπου. Συμπεριφέρεται σαν τον Τοτό, που μετέτρεψε τα πορτοκάλια σε φρούτα και τα μήλα σε φρούτα και, στη συνέχεια, εφάρμοσε σωστά τον τελεστή που πράγματι απαιτεί από τους τελεστέους του να είναι του ίδιου τύπου.

Οι μετατροπές τύπων μπορεί να είναι είτε υπονοούμενες (implicit conversions), οπότε εκτελούνται αυτόματα από το σύστημα, είτε να προσδιορίζονται ρητά από τον προγραμματιστή (explicit conversions).

5.3.1 Υπονοούμενες μετατροπές

Οι υπονοούμενες μετατροπές διευκολύνουν τη δουλειά του προγραμματιστή, ο οποίος όμως, σε κάθε περίπτωση, πρέπει να γνωρίζει τις συνέπειες μιας τέτοιας μετατροπής. Για παράδειγμα, η έκφραση `3.0 + 1 / 2` δεν δίνει τιμή `3.5`, όπως πιθανόν θα αναμένατε, αλλά 3.0, γιατί η διαίρεση ακεραίων `1 / 2` δίνει ακέραιο αποτέλεσμα 0.

Η C βασίζει τη διαδικασία της αυτόματης μετατροπής, στον κανόνα ο οποίος ορίζει ότι **ο στενότερος τύπος μετατρέπεται στον ευρύτερο** χωρίς να υπάρχει απώλεια πληροφορίας. Ο κανόνας αυτός βασίζεται στο γεγονός ότι όλοι οι τύποι της γλώσσας ταξινομούνται ανάλογα με το μέγεθος της μνήμης που απαιτούν για αποθήκευση

(`char` < `int` < `long` < `float` < `double`).

Έτσι, αν `int i;` και `float f;` η έκφραση `f = i = 3.3` θα αποδώσει τιμή 3 στη μεταβλητή `i` και, στη συνέχεια, 3.0 στη μεταβλητή `f`. Επιπλέον, όλοι οι C μεταγλωττιστές όταν υπολογίζουν αριθμητικές εκφράσεις, μετατρέπουν αυτόματα τον τύπο `char` σε `int` και τον `float` σε `double`.

Για περισσότερες πληροφορίες για το σημαντικό αυτό θέμα των μετατροπών

μπορείτε να ανατρέξετε στην ενότητα 3.6 του [Darnel 91], ενώ πολλές ασκήσεις μπορείτε να βρείτε στο 3.2.4 του [Corriveau 98] και το 5.7 του [Rojiani 96]. Ένα καλό παράδειγμα που καλύπτει όλες σχεδόν της αυτόματες μετατροπές δίνεται στη σελίδα 126 του [King '96].

Άσκηση για παραπέρα εξάσκηση 5.1

Υπολογίστε τις τιμές των εκφράσεων `2.56 + 3/2` και `2.56 + 3.0 /2`. Περιγράψτε τις μετατροπές που λαμβάνουν χώρα για τον υπολογισμό της κάθε τιμής. Στη συνέχεια, αναπτύξτε ένα πρόγραμμα που θα τις υπολογίζει και θα τις τυπώνει στην οθόνη. Τέλος, μπορείτε να ανατρέξετε στο παράδειγμα 5.11 ενότητα 5.7 του [Rojiani '96] για να επιβεβαιώσετε τα αποτελέσματά σας.

5.3.2 Ρητές μετατροπές

Εκτός των παραπάνω υπονοούμενων μετατροπών, η C μας επιτρέπει να κάνουμε ρητές μετατροπές τιμών ενός τύπου σε ένα διαφορετικό τύπο. Η ρητή αυτή μετατροπή είναι γνωστή σαν casting και εκτελείται με μια κατασκευή γνωστή με το όνομα **cast**. Για να μετατρέψετε τον τύπο μίας έκφρασης, τοποθετήστε τον τύπο προορισμού, περικλείοντάς τον μέσα σε παρενθέσεις, μπροστά από την έκφραση. Για παράδειγμα, στην έκφραση `j = (float)2`, μετατρέπεται ο ακέραιος 2 σε `float` πριν εκχωρηθεί στη μεταβλητή j. Βέβαια, αν η j είναι ακέραιος, ο μεταγλωττιστής αυτόματα θα μετατρέψει τον `float` σε `int` και μετά θα τον εκχωρήσει στην `j`.

Άσκηση Αυτοαξιολόγησης 5.5

Θεωρήστε την πρόταση `res1 = 2.56 + 3/2;` Ποια από τις παρακάτω μετατροπές δίνει το ίδιο αποτέλεσμα με αυτό της `res2 = 2.56 + 3.0 / 2;`

Α. `res1 = (float)(2.56 + 3/2);`

Β. `res1 = 2.56 + (float)(3/2);`

Γ. `res1 = 2.56 + (float)3/2;`

5.4 Πρόταση

Η πρόταση είναι μία πλήρης *εντολή* (command) προς τον υπολογιστή και προσδιορίζει την εκτέλεση συγκεκριμένου έργου. Το Ελληνικό ερωτηματικό (;) προσδιορίζει το τέλος κάθε πρότασης για την C. Με τον περίεργο αυτό κανόνα μπορούμε να μετατρέψουμε κάθε έκφραση της C σε πρόταση παραθέτοντας απλά το ; στο τέλος της έκφρασης.

Οι προτάσεις ανάλογα με το έργο που επιτελούν διακρίνονται σε κατηγορίες, οι σημαντικότερες των οποίων δίνονται στον πίνακα 5.3 με αντίστοιχα παραδείγματα.

Πίνακας 5.3
Σημαντικότερες κατηγορίες προτάσεων.

Κατηγορία πρότασης	Παράδειγμα
Δήλωσης	`int num;`
κλήσης συνάρτησης	`printf("Hello World");`
ελέγχου ροής	`if (a>b) then a else b;`
ανάθεσης	`num = 21;`
μηδενική	`;`

5.4.1 Σύνθετη πρόταση

Οι γλώσσες προγραμματισμού δίνουν συνήθως τη δυνατότητα ομαδοποίησης προτάσεων για τη δημιουργία κατασκευών υψηλότερης αφαιρετικότητας, όπως το module, η συνάρτηση, η μέθοδος κ.λπ.

Η ομαδοποίηση αυτή επιτυγχάνεται περικλείοντας σε ειδικά σύμβολα το σύνολο των προτάσεων που αποτελούν την υψηλότερης αφαιρετικότητας κατασκευή. Η Pascal, για παράδειγμα, χρησιμοποιεί το σύμβολο `begin` πριν την πρώτη πρόταση και το `end` μετά την τελευταία. Η C ομαδοποιεί τις προτάσεις περικλείοντας αυτές με τα σύμβολα `{` και `}`. Η κατασκευή αυτή αποτελεί για την C μια σύνθετη πρόταση. Οι προτάσεις

```
int num;                num = max(12,13);
int count = 1;          printf("Hello");
count = count + 1;
```

είναι απλές, ενώ οι παρακάτω είναι σύνθετες.

α)
```
{
  int num;
  int count = 1;
  }
```

β)
```
{
  num = max(12,13);
  printf("num = %d\n", num);
  }
```

Στη συνέχεια και όπου χρησιμοποιούμε τον όρο ***Πρόταση*** θα αναφερόμαστε σε απλή ή σύνθετη πρόταση, εκτός εάν ορίζεται διαφορετικά.

5.4.2 Προτάσεις Προεπεξεργαστή

Μια ειδική κατηγορία προτάσεων που θα συναντήσουμε σε C προγράμματα, είναι οι προτάσεις του προεπεξεργαστή[1]. Στην κατηγορία αυτή ανήκει η πρόταση συμπερίληψης που ήδη συναντήσαμε και έχει την μορφή

`#include` <προσδιορισμός αρχείου>

Η πρόταση δίνει εντολή στον προεπεξεργαστή να εισάγει στη θέση της, το αρχείο το οποίο η πρόταση προσδιορίζει.

Μια εξίσου σημαντική εντολή που χρησιμοποιείται σε μεγάλη έκταση είναι η `define`. Η απλή χρήση της `define` είναι η ανάθεση συμβολικού ονόματος σε σταθερά του προγράμματος. Αυτό γίνεται με την παρακάτω σύνταξη:

```
#define        PI      3.141592654
```

Έχοντας κάνει αυτή τη δήλωση μπορούμε να χρησιμοποιούμε στις εκφράσεις μας αντί της σταθεράς 3.141592654 το όνομα `PI`. Έτσι, η έκφραση υπολογισμού της επιφάνειας του κύκλου θα έχει την μορφή `area = PI * r * r;`

Ο προεπεξεργαστής θα αναλάβει να αντικαταστήσει, πριν τη μεταγλώττιση, το συμβολικό όνομα `PI` με την πραγματική τιμή. Αυτό μας δίνει αφενός μεν ένα απλό τρόπο αναφοράς σε σταθερές, αφετέρου δε μας διευκολύνει στις περιπτώσεις αλλαγής της τιμής της σταθεράς. Πολύ εύκολα αλλάζουμε την τιμή της σταθεράς σε 3.14 μόνο στην πρόταση δήλωσης και ο προεπεξεργαστής θα αναλάβει να κάνει την αλλαγή σε όλο τον πηγαίο κώδικα. Αυτή η τεχνική, χρησιμοποιείται σε μεγάλη έκταση και υποστηρίζει την εφαρμογή του ελεγχόμενου πλεονασμού (controlled redundancy).

Μια σύντομη αναφορά στις εντολές του προεπεξεργαστή μπορείτε να βρείτε στην ενότητα 2.7 του [Rojiani '96] ή στην 4.11 του [Kernighan '88], ενώ για μια πιο αναλυτική και σε βάθος αναφορά ανατρέξτε στο κεφάλαιο 14 του [King '96] ή στο κεφάλαιο 10 του [Darnell 1991]. Προσέξτε ιδιαίτερα τη χρήση της define για μακρο–αντικατάσταση με ορίσματα.

5.5 Οι τελεστές της C

Η C διαθέτει 6 κατηγορίες τελεστών. Οι κατηγορίες αυτές είναι: αριθμητικοί,

[1] Ο προεπεξεργαστής της C είναι ένα πρόγραμμα που επεξεργάζεται τον πηγαίο κώδικα πριν από τον μεταγλωττιστή. Αναγνωρίζει ορισμένες εντολές και προκαλεί ένα σύνολο από τροποποιήσεις στον πηγαίο κώδικα πριν αρχίσει το έργο της μεταγλώττισης.

ανάθεσης, συσχετιστικοί, λογικοί, δυαδικών ψηφίων και ειδικοί τελεστές. Στη συνέχεια, γίνεται μια σύντομη αναφορά στους περισσότερους από αυτούς. Για περισσότερες πληροφορίες θα πρέπει να ανατρέξετε στην βιβλιογραφία.

5.5.1 Αριθμητικοί τελεστές

Στους αριθμητικούς τελεστές περιλαμβάνονται οι γνωστοί `+`, `–`, `*`, `/` και ο τελεστής υπολοίπου `%`. Ο τελεστής – εμφανίζεται είτε σαν δυαδικός τελεστής, οπότε και αναπαριστά τη διεργασία της αφαίρεσης, είτε σαν μοναδιαίος τελεστής, οπότε αναπαριστά τη διεργασία του πολλαπλασιασμού του μοναδικού του τελεστέου με το –1.

Όλοι οι αριθμητικοί τελεστές έχουν προσεταιριστικότητα από αριστερά προς τα δεξιά. Οι τελεστές `+` και – δέχονται σαν τελεστέους ακέραιους, πραγματικούς και δείκτες, οι * και `/` δέχονται ακέραιους και πραγματικούς ενώ ο `%` μόνο ακέραιους. Μια συνήθης χρήση του τελεστής `%` είναι για τον έλεγχο της ροής του προγράμματος. Η έκφραση `count % 3` έχει ψευδή τιμή για `count` πολλαπλάσιο του 3 και αληθή για κάθε άλλη τιμή της `count`. Την τιμή της έκφρασης χρησιμοποιεί μια πρόταση διακλάδωσης, όπως θα δούμε σε επόμενο κεφάλαιο, για να κατευθύνει τη ροή εκτέλεσης του προγράμματος.

Στην κατηγορία των αριθμητικών τελεστών ανήκουν και **οι μοναδιαίοι τελεστές αύξησης και μείωσης,** καθώς και οι τελεστές ανάθεσης. Και οι δύο κατηγορίες χρησιμοποιούνται σε μεγάλη έκταση δίνοντας πιο συμπαγή κώδικα. Σε ένα πρόγραμμα, η αύξηση ή ελάττωση της τιμής μιας μεταβλητής είναι πολύ συχνό φαινόμενο. Η C για να διευκολύνει το έργο αυτό διαθέτει δύο μοναδιαίους τελεστές `++` και – –, για αύξηση και μείωση αντίστοιχα κατά 1 της τιμής της μεταβλητής στην οποία εφαρμόζονται. Οι τελεστές αυτοί εφαρμόζονται μόνο σε μεταβλητές και χρησιμοποιούνται είτε σαν προπορευόμενοι (εφαρμόζονται πριν την μεταβλητή, π.χ. `++count`) είτε σαν παρελκόμενοι (εφαρμόζονται μετά τη μεταβλητή, π.χ. `count++`). Παρότι και στις δύο περιπτώσεις η τιμή της μεταβλητής αυξάνεται ή μειώνεται αντίστοιχα κατά ένα, υπάρχει μια σημαντική διαφορά που έχει να κάνει με το πότε εκτελείται η πράξη της αύξησης ή μείωσης.

Η αύξηση ή ελάττωση γίνεται, στην περίπτωση που ο τελεστής είναι προπορευόμενος, πριν να χρησιμοποιηθεί η τιμή της μεταβλητής στον υπολογισμό της τιμής της έκφρασης, ενώ γίνεται μετά, στην περίπτωση που ο τελεστής είναι παρελκόμενος. Το σχήμα 5.2, δίνει τις τιμές των μεταβλητών `x` και `y` μετά την εκτέλεση κάθε μιας από τις διαδοχικές προτάσεις της αριστερής του στήλης

πρόταση	τιμή x	τιμή y
`int x = 10, y = 20;`	10	20
`++x;`	11	20
`y = – –x;`	10	10
`y = x– – + y;`	9	20
`y = y – x++;`	10	11

Σχήμα 5.2
Προπορευόμενοι και παρελκόμενοι μοναδιαίοι τελεστές αύξησης και μείωσης.

Ενδιαφέρουσα είναι η επίδραση των τελεστών μοναδιαίας αύξησης ή μείωσης σε δείκτη, όπου έχουν σαν αποτέλεσμα την αύξηση ή ελάττωση του δείκτη, όχι κατά 1 byte, αλλά κατά τόσα ώστε αυτός να δείχνει στο επόμενο ή προηγούμενο στοιχείο, αντίστοιχα. Για παράδειγμα, αν έχουμε τις δηλώσεις

```
int temp[20];
int *temp_ptr = temp; /* η temp_ptr δείχνει το πρώτο στοιχείο */
                      /* του πίνακα temp */
```

η έκφραση `temp_ptr++` έχει σαν αποτέλεσμα την αύξηση της τιμής του δείκτη ώστε να δείχνει το δεύτερο στοιχείο του πίνακα. Έτσι, αν το πρώτο στοιχείο ήταν στη θέση 1000, η νέα τιμή του δείκτη δεν θα είναι 1001 αλλά 1002 ή 1004, ανάλογα με τον αριθμό των bytes που χρησιμοποιεί το σύστημά σας για την αποθήκευση του ακεραίου.

Άσκηση Αυτοαξιολόγησης 5.6

Προσδιορίστε την τιμή των x και z μετά την εκτέλεση κάθε μίας από τις παρακάτω προτάσεις, θεωρώντας πριν την εκτέλεση της κάθε πρότασης, σαν τιμές των x και y, τα 10 και 20 αντίστοιχα

α) `z = ++x + y;` β) `z = – –x + y;`

γ) `z = x++ + y;` δ) `z = x– – + y;`

5.5.2 Τελεστές ανάθεσης

Εκτός από τον απλό τελεστή ανάθεσης = τον οποίο ήδη συναντήσαμε, η C διαθέτει για κάθε αριθμητικό τελεστή, έναν αντίστοιχο σύνθετο τελεστή ανάθεσης που αποτελείται από τον αριθμητικό τελεστή ακολουθούμενο από τον τελεστή ανάθεσης. Έτσι, έχουμε τους τελεστές +=, –=, *= και /= που αντιστοιχούν στους βασικούς αριθμητικούς τελεστές +, –, * και /.

Οι σύνθετοι τελεστές ανάθεσης είναι δυαδικοί και η διεργασία που εκτελούν συνί-

σταται στην εκτέλεση της διεργασίας που ορίζει ο αριθμητικός τελεστής που περιλαμβάνουν πάνω στους δύο τελεστέους και η ανάθεση στη συνέχεια του αποτελέσματος στον αριστερό τελεστέο. Έτσι, η πρόταση `x *= 10;` εκτελεί την πράξη του πολλαπλασιασμού μεταξύ των x και 10 και εκχωρεί το αποτέλεσμα στο `x`. Είναι φανερό δε ότι αντιστοιχεί στην πρόταση `x = x * 10;` Η έκφραση `x *= y +1` ισοδυναμεί με την `x = x * (y + 1)`. Προσέξτε! όχι με την `x = x * y + 1`.

Τελεστές ανάθεσης δημιουργούν και οι τελεστές διαχείρισης δυαδικών ψηφίων (bitwise operators): `>>= <<= &= ^= |=`

Οι τελεστές ανάθεσης μαζί με τους τελεστές αύξησης και μείωσης αναφέρονται σαν **παρενεργοί τελεστές** (side effect operators), γιατί γίνονται αιτία δημιουργίας παρενεργειών (side effects). Οι παρενέργειες αυτές έχουν σαν αποτέλεσμα την απροσδιόριστη συμπεριφορά του συστήματος ως προς τον τρόπο υπολογισμού της τιμής της μεταβλητής i σε εκφράσεις όπως οι

```
i = v[i++];          /* η συμπεριφορά είναι απροσδιόριστη */
i = ++i + 1;         /* η συμπεριφορά είναι απροσδιόριστη */
```

Εκφράσεις ανάλογης μορφής συνίσταται να αποφεύγονται. Στα πλαίσια 5.2 και 5.4 του [Darnel 91] μπορείτε να βρείτε μια πιο αναλυτική περιγραφή των παρενεργειών που οι τελεστές αυτοί δημιουργούν.

Άσκηση Αυτοαξιολόγησης 5.7

Δώστε το αποτέλεσμα της εκτέλεσης των παρακάτω τμημάτων κώδικα

```
int count = 10;
printf("%d %d\n", count, count*count++);
```

και

```
int num, count = 10;
num = count/2 + 5* (1+ count++);
```

Άσκηση για παραπέρα εξάσκηση 5.2

Χρησιμοποιώντας τον πίνακα προτεραιότητας και προσεταιριστικότητας των τελετών της C υπολογίστε την τιμή της έκφρασης

```
a = b += c++ –d + – –e / –f
```

Στη συνέχεια, ανατρέξτε στην ενότητα 4.4 του [King 96] όπου μπορείτε να βρείτε σχόλια για τον τρόπο υπολογισμού της πολύπλοκης αυτής έκφρασης.

5.5.3 Συσχετιστικοί τελεστές

Οι *συσχετιστικοί τελεστές* ή αλλιώς *τελεστές σύγκρισης*, είναι οι `<` `>` `<=` `>=` `==` και `!=`

Από αυτούς οι τέσσερις πρώτοι, σας είναι σίγουρα γνωστοί. Ο == είναι ο τελεστής ελέγχου ισότητας και ο `!=` είναι ο τελεστής διάφορο. Το αποτέλεσμα μιας έκφρασης που περιλαμβάνει τελεστή σύγκρισης, είναι αληθές (true) ή ψευδές (false). Αν και ορισμένες γλώσσες, όπως η Pascal, έχουν Boolean τύπο δεδομένων για την αναπαράσταση αυτών των δύο τιμών, η C αναπαριστά τις τιμές αυτές με ακέραια μεταβλητή, με τη σύμβαση ότι το μηδέν (0) αντιστοιχεί στο ψευδές και κάθε άλλη τιμή στο αληθές. Έτσι, η τιμή μιας συσχετιστικής έκφρασης είναι ακέραια 0 για false και 1 για true.

Προσέξτε, ώστε να αποφεύγετε τη σύγκριση ισότητας μεταξύ πραγματικών αριθμών. Η έκφραση `(1.0/3.0 + 1.0/3.0 + 1.0/3.0) == 1.0` αν και αλγεβρικά σωστή, υπολογίζεται σαν ψευδής στους περισσότερους υπολογιστές. Για μια αναλυτική αναφορά στο θέμα μπορείτε να ανατρέξετε στην ενότητα 5.7 του [Darnel '91].

Άσκηση Αυτοαξιολόγησης 5.8

Αντιστοιχήστε τις παρακάτω εκφράσεις με τις τιμές true και false

`-1 < 0`	**true**	`1!= -1`
`0>1`		`1>= -1`
`0 == 0`	**false**	`1>10`
`100`		`-100`

Άσκηση Αυτοαξιολόγησης 5.9

Υπολογίστε τις τιμές των παρακάτω εκφράσεων:

α) `count = 8` β) `num = = 9`

γ) `num = count + 4` δ) `count = = num - 4`

Με δεδομένο ότι οι `count` και `num` είναι ακέραιες μεταβλητές και πριν από κάθε έκφραση έχουν τιμές 8 και 12 αντίστοιχα.

5.5.4 Λογικοί τελεστές &&, || και !

Οι λογικοί τελεστές AND (`&&`), OR (`||`) και NOT (`!`) χρησιμοποιούνται για να συνενώσουν δύο οι περισσότερες συσχετιστικές εκφράσεις, για τη δημιουργία λογικών εκφράσεων. Για παράδειγμα, αν θέλουμε να ελέγξουμε αν το `y` είναι μεγαλύτερο του `x` και μικρότερο του `z`, δεν μπορούμε να γράψουμε για την C την έκφραση `x < y <z` αλλά θα πρέπει να δημιουργήσουμε δύο συσχετιστικές εκφράσεις, τις `x < y` και `y < z` και να τις συνδυάσουμε με τον λογικό τελεστή AND για τη δημιουργία της λογική έκφρασης

`(x < y) && (y < z)`, η οποία είναι αληθής όταν `x` μικρότερο του `y` και `y` μικρότερο του `z`.

Άσκηση Αυτοαξιολόγησης 5.10

Θυμηθείτε την τεχνική του υπολογισμού περιορισμένης έκτασης και δώστε τις τιμές των `x`, `y`, `z` μετά τον υπολογισμό της τιμής της έκφρασης `(x++ < y— —) && (y— — < ++z)`

για α) `x= 3`, `y = 4` και `z= 4` και
β) `x= 5`, `y = 4` και `z= 4`.

5.5.5 Τελεστές Μνήμης

Οι *τελεστές μνήμης* είναι οι: `& * [ ] . —>`

Τους τρεις πρώτους τους συναντήσαμε ήδη στους δείκτες και τους πίνακες. Τους άλλους δύο θα δούμε στο κεφάλαιο των δομών.

5.5.6 Ο τελεστής μετατροπής τύπου

Ο *τελεστής μετατροπής τύπου* ή **cast** τελεστής όπως αποκαλείται, είναι μοναδιαίος και έχει τη μορφή (τύπος δεδομένων). Τοποθετείται μπροστά από μια έκφραση για να μετατρέψει την τιμή της στον περικλειόμενο σε παρενθέσεις τύπο. Είναι ο τελεστής που υποστηρίζει τη ρητή μετατροπή τύπων που ήδη αναφέραμε στην υποενότητα 5.3.2.

5.5.7 Τελεστής sizeof

Ο *τελεστής* `sizeof` είναι μοναδιαίος και δρα πάνω σε δύο τύπους τελεστέων: α) σε έκφραση π.χ. `sizeof(x+y)` και

β) σε τύπο δεδομένων π.χ. `sizeof(int)`.

Σε κάθε περίπτωση, επιστρέφει τον αριθμό των bytes που η τιμή της έκφρασης ή ο τύπος δεδομένων καταλαμβάνει στη μνήμη. Αξίζει να σημειωθεί ότι το σύστημα δεν υπολογίζει την τιμή της έκφρασης και έτσι, πιθανή ύπαρξη παρενεργών τελεστών δεν δημιουργεί προβλήματα.

Για παράδειγμα η `printf ("%d\n", sizeof(int));`

τυπώνει τον αριθμό των bytes που απαιτούνται για την αποθήκευση του τύπου `int`.

Άσκηση για παραπέρα εξάσκηση 5.3

Γράψτε ένα πρόγραμμα που θα τυπώνει στην οθόνη το μέγεθος σε bytes που καταλαμβάνει ο κάθε ένας από τους τύπους `char`, `short`, `int`, `float`, και `double` στο σύστημά σας.

5.5.8 Τελεστές διαχείρισης bits

Οι *τελεστές δυαδικών ψηφίων* επιτρέπουν πρόσβαση και διαχείριση των bits της μνήμης που καταλαμβάνει ένα αντικείμενο. Είναι ιδιαίτερα χρήσιμοι για προγραμματισμό χαμηλού επιπέδου (low level programming). Η χρήση τους απαιτεί πολύ καλή εξοικείωση με τη γλώσσα. Μια καλή αναφορά μπορείτε να βρείτε στο [Darnell 91] ενώ για την χρήση τους σε low level programming μπορείτε να ανατρέξετε στο κεφάλαιο 20 του [King 96].

5.5.9 Υποθετικός Τελεστής

Ο υποθετικός τελεστής `(? : )` αποτελείται από δύο σύμβολα, ανήκει στην κατηγορία των τελεστών, οι οποίοι αποτελούνται από συνδυασμό συμβόλων και δεν ακολουθούν καμία από τις postfix, prefix ή infix σημειολογίες. Όταν τα σύμβολα ή οι λέξεις του τελεστή είναι διάσπαρτα στους τελεστέους στους οποίους εφαρμόζεται ο τελεστής, λέμε ότι ο τελεστής είναι σε μικτή σημειολογία (mixfix notation). Για παράδειγμα, στην έκφραση `x > z ? x : z` τα σύμβολα `?` και : του υποθετικού τελεστή παρεμβάλλονται μεταξύ των x > z, x και z.

Η έκφραση που σχηματίζει ο υποθετικός τελεστής έχει τη μορφή

εκφρ1 ? εκφρ2 : εκφρ3

Η τιμή της είναι η τιμή της εκφρ2, εάν η εκφρ1 είναι αληθής, αλλιώς είναι η τιμή της εκφρ3. Η εκφρ1 που αποτελεί τη συνθήκη ελέγχου, πρέπει να είναι

βαθμωτού τύπου. Μόνο μία από τις δύο εκφράσεις υπολογίζεται ανάλογα με την τιμή της εκφρ1.

Έτσι, η έκφραση `x>z ? x : z` έχει τιμή `x` εάν το `x > z` είναι αληθές, διαφορετικά έχει τιμή z.

Δραστηριότητα 5.1

Γράψτε ένα πρόγραμμα που θα ζητά από τον χρήστη δύο ακέραιους αριθμούς και, στη συνέχεια, θα βρίσκει τον μεγαλύτερο και θα τον τυπώνει. Τροποποιήστε το ώστε να τυπώνει τον μικρότερο. Τέλος, δώστε μια έκδοση για τρεις αριθμούς. Το δικό μας σχόλιο θα βρείτε στο τέλος του βιβλίου.

Σύνοψη

Η γλώσσες προγραμματισμού χρησιμοποιούν τους τελεστές για να αναπαραστήσουν βασικές αριθμητικές, συσχετιστικές, λογικές και άλλες διεργασίες. Οι τελεστές συνδυάζονται με τελεστέους για να σχηματίσουν εκφράσεις, οι οποίες με τη σειρά τους δημιουργούν τις προτάσεις που συνθέτουν ένα πρόγραμμα. Σαν τελεστέοι χρησιμοποιούνται μεταβλητές, σταθερές, και κλήσεις συναρτήσεων. Οι τιμές αυτών υφίστανται τη διεργασία που αναπαριστά ο τελεστής και το αποτέλεσμα αποτελεί την τιμή της έκφρασης. Όταν σε μια έκφραση εμφανίζονται περισσότεροι του ενός τελεστές, η σειρά εκτέλεσης των διεργασιών που αναπαριστούν ορίζεται από την προτεραιότητα και την προσεταιριστικότητα των τελεστών. Ο μεταγλωτιστής για τον υπολογισμό της τιμής μιας έκφρασης δημιουργεί ένα δένδρο αφηρημένης σύνταξης με κόμβους τους τελεστές και φύλλα τους τελεστέους. Αν και ο υπολογισμός γίνεται συνήθως με την εφαρμοστική σειρά, ορισμένες γλώσσες υποστηρίζουν και τον υπολογισμό περιορισμένης έκτασης για τους λογικούς τελεστές and και or. Η C διαθέτει πλούσιο σύνολο τελεστών, πλουσιότερο αυτού της Pascal, για τους οποίους, επιπλέον, επιτρέπει την ανάμειξη των τύπων της σχεδόν χωρίς περιορισμούς. Οι αυτόματες αλλά και οι ρητά οριζόμενες από τον προγραμματιστή μετατροπές τύπων επιτρέπουν τη δημιουργία σύνθετων αλλά πολύ αποδοτικών εκφράσεων, καθιστώντας τη γλώσσα ισχυρότατο εργαλείο στα χέρια του προγραμματιστή.

Βιβλιογραφία

[Corriveau ’98]

Corriveau Jean Paul, «*A step–by–step Guide to C Programming*», Prentice Hall.

[Darnell 1991]

Darnell P., Margolis P. «*C: A Software Engineering Approach*», Springer–Verlag, New York.

[Kernighan ’88]

Η δεύτερη έκδοση του βιβλίου «*The C Programming Language*» κυκλοφόρησε το 1988 καλύπτοντας πλέον την ANSI C.

[King ’96]

King K. N., «*C Programming: A modern approach*», W.W.Norton & Company, Inc.

[Rojiani ’96]

Rojiani K. B., «*Programming in C with numerical methods for Engineers*», Prentice–Hall.

ΚΕΦΑΛΑΙΟ 6

Αφαιρετικότητα στις Διεργασίες

Σκοπός

Σκοπός του κεφαλαίου είναι να εισάγει τη συνάρτηση σαν μηχανισμό των γλωσσών προγραμματισμού για την υλοποίηση της αφαιρετικότητας στις διεργασίες.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *δώσετε τον ορισμό της αφαιρετικότητας (abstraction),*
- *αναφέρετε τις δύο βασικές κατηγορίες αφαιρετικότητας,*
- *περιγράψετε τα επί μέρους τμήματα μιας διεργασίας,*
- *να διακρίνετε την υλοποίηση (implementation) από τον τρόπο αναφοράς μιας διεργασίας (interface),*
- *δώσετε τον ορισμό της συνάρτησης,*
- *αναφέρετε 4 τουλάχιστον από τα χαρακτηριστικά που θέλουμε να έχει κάθε τμήμα κώδικα (module),*
- *αναφέρετε 4 τουλάχιστον βασικά πλεονεκτήματα που απορρέουν από τη χρήση συναρτήσεων,*
- *περιγράψετε τι προσδιορίζει το πρωτότυπο συνάρτησης,*
- *αναφέρετε τις τρεις διαφορετικές μορφές με τις οποίες συναντάμε στον πηγαίο κώδικα το όνομα μιας συνάρτησης.*

Έννοιες κλειδιά

- *διεργασία*
- *αφαιρετικότητα στις διεργασίες*
- *αφαιρετικότητα στα δεδομένα*
- *αρθρωτός σχεδιασμός (modular design)*
- *συνάρτηση (function)*
- *υπορουτίνα (procedure)*
- *τυπικά – πραγματικά ορίσματα (formal–actual arguments)*
- *επιστρεφόμενη τιμή (return value)*
- *πρωτότυπο συνάρτησης (function prototype)*
- *ορισμός συνάρτησης*
- *κλήση συνάρτησης*

Εισαγωγικές Παρατηρήσεις

Μια από τις βασικότερες τεχνικές με τις οποίες ο άνθρωπος αντιμετωπίζει την πολυπλοκότητα είναι η αφαιρετικότητα (abstraction). Η αφαιρετικότητα αποτελεί μια απλοποιημένη περιγραφή ή τεκμηρίωση που δίνει έμφαση σε ορισμένα χαρακτηριστικά, ενώ ταυτόχρονα αποσιωπεί άλλα. Η έννοια είναι βασικής σημασίας για τη φιλοσοφία και τα μαθηματικά, ενώ βρίσκει πολύ μεγάλη εφαρμογή στη φάση της ανάλυσης συστημάτων. Διακρίνουμε δύο μορφές αφαιρετικότητας:

α) την αφαιρετικότητα στις διεργασίες (procedural abstraction) και

β) την αφαιρετικότητα στα δεδομένα (data abstraction).

Στο κεφάλαιο αυτό που αποτελείται από τρεις ενότητες, θα ασχοληθούμε με την αφαιρετικότητα στις διεργασίες. Η αφαιρετικότητα στα δεδομένα θα αποτελέσει θέμα άλλου κεφαλαίου. Στην πρώτη ενότητα γίνεται μια αναφορά στην έννοια της αφαιρετικότητας και στον τρόπο που αυτή χρησιμοποιείται στον προγραμματισμό για να υποστηρίξει τη διάκριση μεταξύ τού τι κάνει ένα τμήμα κώδικα και του πώς το κάνει. Η δεύτερη ενότητα κάνει μια πολύ σύντομη αναφορά στο θέμα του αρθρωτού σχεδιασμού και το κεφάλαιο ολοκληρώνεται με την τρίτη ενότητα η οποία εισάγει τη γλωσσική κατασκευή της συνάρτησης.

6.1 Αφαιρετικότητα στις διεργασίες

Στην εισαγωγή του βιβλίου αναφερθήκαμε στη διεργασία ως βασικό δομικό στοιχείο της διαδικασίας ανάπτυξης συστημάτων λογισμικού. Στην καθημερινή μας ζωή αναφερόμαστε συχνά σε διεργασίες με τα ονόματά τους, τα οποία δίνουν έμφαση στο **τι** κάνει η διεργασία και **όχι στο πώς**. Η φράση «φτιάξε ένα νες μέτριο με γάλα» επιτρέπει στο χρήστη να εστιάσει την προσοχή του στο τι πρέπει να γίνει και όχι στο πώς αυτό γίνεται. Έτσι, μπορείτε να πίνετε το νες σας χωρίς να είναι ανάγκη να γνωρίζετε τον τρόπο παρασκευής του.

Μια διεργασία μπορεί να θεωρηθεί αποτελούμενη από τρία επί μέρους τμήματα:

1. το όνομα της διεργασίας

2. τις εισόδους και εξόδους της διεργασίας

3. το σώμα της διεργασίας

Το όνομα χρησιμοποιείται για να προσδιορίσει τη διεργασία (φτιάξε ένα νες). Οι είσοδοι προσδιορίζουν το σύνολο των στοιχείων εισόδου της διεργασίας (μέτριο, με γάλα) και οι έξοδοι το αποτέλεσμα της διεργασίας (ο νες σας). Τέλος, το σώμα της διεργασίας είναι η περιγραφή ενός συνόλου ενεργειών, οι οποίες έχουν ομαδοποιηθεί με στόχο την εκτέλεση συγκεκριμένου έργου.

Ας υποθέσουμε ότι θέλετε ένα νες. Για να αναθέσετε τη διεργασία της κατασκευής του σε κάποια άλλη οντότητα, θα πρέπει να γνωρίζετε τον τρόπο αναφοράς στη διεργασία. Από την άλλη μεριά, η οντότητα στην οποία θα αναθέσετε τη διεργασία, ο μπάρμαν ή ακόμη το αυτόματο μηχάνημα, θα πρέπει να γνωρίζει όχι μόνο τον τρόπο αναφοράς στη διεργασία, ώστε να την διακρίνει από τις πιθανώς πολλές διεργασίες που μπορεί να εκτελέσει, αλλά και το σώμα της διεργασίας, το σύνολο δηλαδή των επιμέρους ενεργειών που πρέπει να εκτελέσει για να φτιάξει το νες.

Αλλά «τι σχέση έχουν αυτά με μας;» θα μου πείτε «είμαστε στην ενότητα γλώσσες προγραμματισμού». Για θυμηθείτε όμως τον ορισμό που δώσαμε στη γλώσσα προγραμματισμού: «μία συστηματική σημειογραφία (notation) με την οποία περιγράφουμε υπολογιστικές διεργασίες». Τι κάνατε κάθε φορά που σας ζητούσα να φτιάξετε ένα πρόγραμμα; Χρησιμοποιούσατε τη γλώσσα προγραμματισμού για να περιγράψετε τα επί μέρους βήματα της διεργασίας που έπρεπε να εκτελέσει ο υπολογιστής για να σας δώσει το ζητούμενο αποτέλεσμα. Περιγράφατε λοιπόν μια διεργασία, μια υπολογιστική διερ-

γασία για την ακρίβεια. Στη συνέχεια δημιουργούσατε τον εκτελέσιμο κώδικα, το σώμα δηλαδή της διεργασίας σε γλώσσα μηχανής, και το αποθηκεύατε κάπου εκεί στο σκληρό δίσκο του μηχανήματός σας. Τι κάνετε όταν θέλετε ο υπολογιστής να εκτελέσει τη διεργασία της εύρεσης του μέγιστου μεταξύ δύο αριθμών; Αναφέρεστε στη διεργασία με το όνομά της (όνομα του αρχείου του προγράμματος), δίνεται τις εισόδους (τους δύο αριθμούς) και, ο υπολογιστής έχοντας το σώμα της διεργασίας, την εκτελεί και σας δίνει το αποτέλεσμα. Το σώμα της διεργασίας το ονομάζουμε *υλοποίηση* (implementation) της διεργασίας, ενώ τα υπόλοιπα (όνομα, είσοδοι, έξοδοι) αποτελούν τον τρόπο αναφοράς στη διεργασία, τη *διεπαφή* (interface) όπως λέμε της διεργασίας.

Ας δούμε, επιπλέον, τι κάνουμε όταν θέλουμε να ζητήσουμε από τον υπολογιστή να μας εμφανίσει ένα αποτέλεσμα στην οθόνη. Αναφερόμαστε απλά σε μια διεργασία, η οποία επειδή είναι πολύ βασική και χρησιμοποιείται πολύ συχνά, αναπαραστάθηκε από τη συνάρτηση `printf` της βασικής βιβλιοθήκης. Το μεν σώμα της `printf` βρίσκεται σε ένα από τα αρχεία .lib της βασικής βιβλιοθήκης της C, ο δε τρόπος αναφοράς της στο αρχείο επικεφαλίδας `stdio.h` το οποίο και φροντίσαμε να συμπεριλάβουμε στο πρόγραμμά μας χρησιμοποιώντας την εντολή `include` του προεπεξεργαστή.

6.2 Αρθρωτός σχεδιασμός

Ο *αρθρωτός σχεδιασμός* (modular design) βασίζει την ανάπτυξη σύνθετων συστημάτων στην τμηματοποίηση των σύνθετων διεργασιών σε επιμέρους απλούστερες διεργασίες, ή αντίστοιχα, την τμηματοποίηση του προγράμματος σε επιμέρους *τμήματα* (modules), συναρτήσεις, όπως ονομάζονται στη C. Τα πλεονεκτήματα του αρθρωτού σχεδιασμού θα δείτε στην αντίστοιχη θεματική ενότητα, μια σύντομη όμως αναφορά σε αυτά μπορείτε να βρείτε στο [Rojiani 96], στο οποίο μπορείτε να δείτε και μια λίστα με τα θεμιτά χαρακτηριστικά ενός module τα οποία εν συντομία αναφέρουμε στη συνέχεια. Ένα module πρέπει:

1. να είναι σχετικά μικρό,
2. να έχει αν είναι δυνατό μια είσοδο και μια έξοδο,
3. να αναπτύσσεται και ελέγχεται σαν αυτόνομη μονάδα,
4. να εκτελεί ένα σαφώς ορισμένο έργο, και
5. να έχει μια καλώς προσδιορισμένη διεπαφή (interface).

Κάθε επιμέρους διεργασία του προγράμματος αναπτύσσεται σαν μια αυτόνομη συνάρτηση και σε ένα τελικό βήμα όλες οι συναρτήσεις ενοποιούνται για να αποτελέσουν το πρόγραμμα. Η συνάρτηση είναι για την C η γλωσσική κατασκευή που αναπαριστά τη διεργασία. Άλλες κατασκευές που χρησιμοποιούνται για αναπαράσταση διεργασιών είναι η υπορουτίνα (subroutine) και η λειτουργία (operation)[1]. Στη διαφορά μεταξύ συνάρτησης και υπορουτίνας θα αναφερθούμε στο κεφάλαιο της Pascal, η οποία υποστηρίζει και τις δύο γλωσσικές κατασκευές. Όσο για τη λειτουργία, θα δούμε πώς αυτή αποτελεί βασικό χαρακτηριστικό του αντικειμενοστρεφούς προγραμματισμού.

6.3 Συναρτήσεις

Η C υποστηρίζει πλήρως τον αρθρωτό σχεδιασμό με τη γλωσσική κατασκευή που ονομάζεται *συνάρτηση*. Η συνάρτηση είναι μία αυτόνομη μονάδα κώδικα σχεδιασμένη να επιτελεί συγκεκριμένο έργο. Βοηθά στην αποφυγή της επανάληψης, στην αύξηση της επαναχρησιμοποίησης, στη βελτίωση της αναγνωσιμότητας του κώδικα, καθώς και στη βελτίωση της διαδικασίας συντήρησης.

Κάθε C πρόγραμμα αποτελείται από μια ή περισσότερες συναρτήσεις και περιέχει οπωσδήποτε τη συνάρτηση `main` από την οποία, όπως ήδη γνωρίζετε, αρχίζει η εκτέλεση του προγράμματος.

Το όνομα μιας συνάρτησης το συναντάμε σε ένα πρόγραμμα σε προτάσεις τριών διαφορετικών μορφών:

1. πρόταση δήλωσης της συνάρτησης
2. πρόταση ορισμού της συνάρτησης και
3. πρόταση κλήσης της συνάρτησης.

6.3.1 Δήλωση συνάρτησης

Η πρόταση δήλωσης, η ύπαρξη της οποίας είναι απαραίτητη πριν από τη χρήση της συνάρτησης, προσδιορίζει τη διεπαφή της συνάρτησης, με άλλα λόγια, τον τρόπο αναφοράς στη συνάρτηση. Η δήλωση συνάρτησης έχει τη παρακάτω μορφή:

<τύπος> <όνομα συνάρτησης>(<τύπος1> [<όνομα παραμέτρου1>], ...,<τύποςN> [<όνομα παραμέτρουN>]);

Η δήλωση ή το πρωτότυπο της συνάρτησης (function prototype), όπως ονο-

[1] Η οποία είναι γνωστή και σαν μέθοδος (method).

μάζει τη δήλωση η ANSI C, προσδιορίζει τον τύπο της επιστρεφόμενης τιμής, το όνομα της συνάρτησης, τον αριθμό και τον τύπο των παραμέτρων και, προαιρετικά, τα ονόματα με τα οποία το σώμα της συνάρτησης αναφέρεται στις εισόδους που δέχεται. Το όνομα της συνάρτησης, συνοδεύεται είτε από μια λέξη κλειδί (procedure ή function για την Pascal), είτε από έναν τελεστή (τον τελεστή ( ) για την C). Οι παράμετροι ή αλλιώς τα *τυπικά ορίσματα* (formal arguments), είναι το εμφανές μέρος του μηχανισμού της γλώσσας που υποστηρίζει το πέρασμα των εισόδων σε μια συνάρτηση[1]. Αντίστοιχα, η επιστρεφόμενη τιμή είναι το μέρος του μηχανισμού που υποστηρίζει την επιστροφή του αποτελέσματος. Έτσι, για παράδειγμα, η δήλωση

```
int max(int a, int b);
```

προσδιορίζει μια συνάρτηση με όνομα `max`, που δέχεται σαν είσοδο δύο ακέραιους αριθμούς και έχει σαν έξοδο ακέραιο. Εάν η συνάρτηση δεν επιστρέφει τιμή χρησιμοποιείται για τον προσδιορισμό του τύπου της επιστρεφόμενης τιμής η λέξη κλειδί `void`. Έτσι, η συνάρτηση draw_circle που ζωγραφίζει ένα κύκλο με κέντρο το σημείο (x, y) και ακτίνα r και δεν επιστρέφει καμία τιμή έχει σαν δήλωση την

```
void draw_circle(double x, double y, double r);
```

Άσκηση Αυτοαξιολόγησης 6.1

Δώστε το πρωτότυπο συνάρτησης που δέχεται σαν εισόδους τρεις ακεραίους αριθμούς, βρίσκει τον μεγαλύτερο και τον επιστρέφει.

Άσκηση Αυτοαξιολόγησης 6.1

Δώστε τη δήλωση συνάρτησης η οποία θα υπολογίζει το x^n, όπου x πραγματικός αριθμός και n ακέραιος.

Άσκηση Αυτοαξιολόγησης 6.1

Δώστε τη δήλωση συνάρτησης που θα βρίσκει και επιστρέφει τη διεύθυνση της πρώτης εμφάνισης ενός χαρακτήρα σε αλφαριθμητικό.

[1] Πολλές φορές χρησιμοποιούνται και για επιστροφή εξόδων.

6.3.2 Κλήση συνάρτησης

Για να ενεργοποιήσουμε μια συνάρτηση την καλούμε. Έχετε ήδη καλέσει τις συναρτήσεις `printf` και `scanf` της βασικής βιβλιοθήκης. Η κλήση μιας συνάρτησης που δεν επιστρέφει τιμή αποτελείται από το όνομα της συνάρτησης ακολουθούμενο από τη λίστα των παραμέτρων, τα οποία καλούνται *πραγματικά ορίσματα* (actual arguments), έχει δε τη μορφή αυτόνομης πρότασης:

<όνομα συνάρτησης>(όρισμα1, όρισμα2, ..., όρισμα N);

Τα πραγματικά ορίσματα χωρίζονται με κόμμα και περικλείονται σε παρενθέσεις, αντιστοιχούν δε ένα προς ένα στις παραμέτρους (τυπικά ορίσματα) της συνάρτησης. Οι παρενθέσεις είναι απαραίτητες ακόμη και όταν δεν υπάρχουν ορίσματα, για να γνωρίσουν στον μεταγλωττιστή ότι το όνομα είναι συνάρτηση και όχι μεταβλητή. Τα πραγματικά ορίσματα μπορεί να είναι σταθερές, μεταβλητές ή ακόμη και εκφράσεις, αλλά σε κάθε περίπτωση πρέπει να είναι ίδιου τύπου με τα αντίστοιχα τυπικά. Στην πρόταση

```
draw_circle(a/2.0, 2.0*b, c/3.0);
```

υπολογίζονται πρώτα οι εκφράσεις και, στη συνέχεια, ο έλεγχος περνάει στη συνάρτηση `draw_circle`, ενώ ταυτόχρονα αποδίδονται οι τιμές που υπολογίστηκαν στα τυπικά ορίσματα `x`, `y` και r.

Αντίθετα, οι συναρτήσεις οι οποίες επιστρέφουν τιμές αποτελούν τελεστέους σε σύνθετες εκφράσεις όπως αυτές των παρακάτω προτάσεων:

```
max_num = max(num1,num2);
result = num1 + max(num2,num3);
printf("ο μεγαλύτερος αριθμός είναι ο %d\n", max(num1,num2));
```

Δραστηριότητα 6.1

Γράψτε ένα πρόγραμμα στο οποίο θα καλείτε τις συναρτήσεις των οποίων τις δηλώσεις δώσατε στις ασκήσεις Αυτοαξιολόγησης 6.1. 6.2 και 6.3. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

6.3.3 Ορισμός συνάρτησης

Το σώμα των συναρτήσεων `printf` και `scanf` της βασικής βιβλιοθήκης βρίσκεται σε ένα από τα αρχεία .lib που την αποτελούν. Ο συνδέτης θα το εντοπίσει στη διάρκεια της σύνδεσης και θα το ενσωματώσει στον εκτελέσιμο κώδικα του προγράμματός μας. Αντίθετα, για κάθε δική μας συνάρτηση, θα πρέπει να ορίσουμε το σώμα της. Ο ορισμός μιας συνάρτησης αποτελείται

από το πρωτότυπο της συνάρτησης ακολουθούμενο από το σύνολο των προτάσεων που επιτελούν το έργο της συνάρτησης. Σαν παράδειγμα δίνεται ο ορισμός της συνάρτησης `area` που υπολογίζει την επιφάνεια ορθογωνίου.

```
float area(float width, float height)
{
double result;
result = width * height;
return(result);
}
```

Με τα ονόματα `width` και `height` τα οποία είναι τα τυπικά ορίσματα της συνάρτησης, ο προγραμματιστής έχει τη δυνατότητα να αναφέρεται μέσα από το σώμα της συνάρτησης στις εισόδους της. Τα τυπικά ορίσματα χρησιμοποιούνται σε εκφράσεις όπως ακριβώς οι μεταβλητές, με τον περιορισμό ότι μπορούν να χρησιμοποιηθούν μόνο μέσα στο σώμα της συνάρτησης, έχουν, όπως θα δούμε στη συνέχεια, τοπική εμβέλεια. Ο ίδιος ακριβώς περιορισμός ισχύει και για τη μεταβλητή `result`, η οποία δηλώνεται μέσα στο σώμα της συνάρτησης και ονομάζεται *τοπική μεταβλητή* (local variable) σε διάκριση από τις μεταβλητές που δηλώνονται έξω από το σώμα συνάρτησης και ονομάζονται *καθολικές* ή *γενικές μεταβλητές* (global variables).

Τα τυπικά ορίσματα αποτελούν τον έναν από τους δύο τρόπους με τον οποίο μπορεί να επικοινωνεί η καλούσα συνάρτηση με την καλούμενη. Ο δεύτερος τρόπος (τον οποίο θα πρέπει γενικά να αποφεύγετε να χρησιμοποιείτε) είναι η κοινή χρήση γενικών μεταβλητών. Στο θέμα όμως αυτό θα αναφερθούμε αναλυτικότερα σε επόμενο κεφάλαιο.

Μετά την ολοκλήρωση και της τελευταίας πρότασης του σώματος της συνάρτησης, ο έλεγχος επιστρέφει στο σημείο της κλήσης της. Εναλλακτικά, μπορούμε να μεταφέρουμε τον έλεγχο στο σημείο κλήσης από οποιοδήποτε σημείο του σώματος χρησιμοποιώντας την εντολή `return`. Η `return` έχει σαν αποτέλεσμα την επιστροφή του ελέγχου αλλά και της τιμής της έκφρασης που την ακολουθεί, στη συνάρτηση που κάλεσε την `area`. Για τον τερματισμό συνάρτησης χωρίς επιστρεφόμενη τιμή χρησιμοποιούμε την `return` χωρίς έκφραση.

Άσκηση Αυτοαξιολόγησης 6.4

Δώστε τον ορισμό της συνάρτησης υπολογισμού του μέγιστου μεταξύ δύο ακέραιων αριθμών.

Άσκηση Αυτοαξιολόγησης 6.5

Η συνάρτηση της οποίας ο πηγαίος κώδικας δίνεται στη συνέχεια, υπολογίζει το εμβαδόν ενός τριγώνου. Εντοπίστε τα τρία λάθη του πηγαίου κώδικα της και διορθώστε τα. Αν δεν τα καταφέρετε ζητήστε τη βοήθεια του μεταγλωττιστή σας.

```
float triangle_area(float base, height)
float ginomeno;
{
ginomeno = base * height;
return(ginomeno/base)
}
```

6.3.4 Επεξήγηση του μηχανισμού κλήσης συνάρτησης

Η επεξήγηση του μηχανισμού κλήσης συνάρτησης γίνεται χρησιμοποιώντας σαν παράδειγμα την πρόταση

```
result = max(max(15,13),17);
```

Σε μια πρόταση ανάθεσης όπως η παραπάνω, υπολογίζεται πρώτα η τιμή του δεξιού τελεστέου και, στη συνέχεια, αυτή αποδίδεται στη μεταβλητή που προσδιορίζει ο αριστερός τελεστέος. Ο δεξιός τελεστέος είναι μία έκφραση που αποτελείται μόνο από την κλήση της συνάρτησης max, με πρώτο πραγματικό όρισμα την έκφραση `max(15,13)` και δεύτερο τη σταθερά 17. Σύμφωνα με το μηχανισμό της C για την κλήση συνάρτησης, υπολογίζεται πρώτα η τιμή των πραγματικών ορισμάτων[1]. Για τον υπολογισμό του ορίσματος `max(15,13)`, που είναι και αυτό έκφραση κλήσης συνάρτησης, αποδίδονται οι τιμές 15 και 13 των πραγματικών ορισμάτων στις παραμέτρους `a` και `b` αντίστοιχα, όπως φαίνεται στο σχήμα 6.1(α), και ο έλεγχος μεταφέρεται στο σώμα της `max`. Το σώμα της max υπολογίζει την έκφραση `(a > b) ? a : b` με τα a και b να είναι 15 και 13, αντίστοιχα. Η πρόταση return μεταφέρει τον έλεγχο εκεί από όπου κλήθηκε η συνάρτηση επιστρέφοντας ταυτόχρονα και την τιμή 15 που υπολόγισε. Η τιμή αυτή αποτελεί την τιμή του πρώτου ορίσματος της δεύτερης κλήσης της `max`. Η δεύτερη αυτή κλήση της max έχει σαν αποτέλεσμα, όπως δείχνει το σχήμα 6.1(β), την απόδοση των τιμών 15 και 17 στις παραμέτρους `a` και `b`, αντίστοιχα, της `max` και, στη συνέχεια, την μεταφορά του ελέγχου στο σώμα της `max` για δεύτερη φορά. Τώρα όμως η

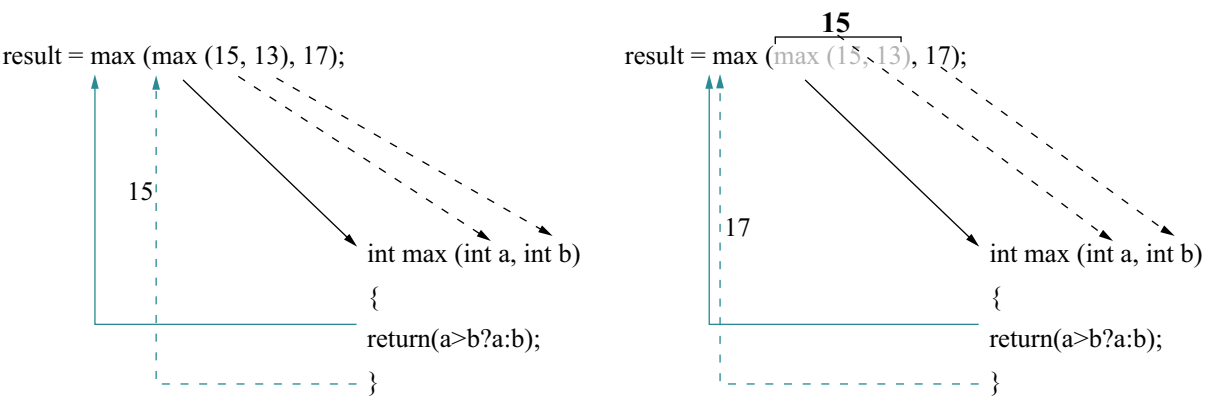
[1] Η ANSI C δεν ορίζει τη φορά υπολογισμού των πραγματικών ορισμάτων.

έκφραση `(a>b) ? a : b`, με τα `a` και `b` να είναι 15 και 17, δίνει 17 το οποίο και επιστρέφεται με την `return` και αποτελεί την τιμή της δεξιού τελεστέου της πρότασης ανάθεσης.

Μια πιο αναλυτική περιγραφή του μηχανισμού κλήσης συνάρτησης μπορείτε να βρείτε στη σελίδα 183 του [Sethi 97] κάτω από τον τίτλο «Procedure Call and Return in C».

6.4 Η βασική βιβλιοθήκη της C

Η βασική βιβλιοθήκη της C ορίζει ένα σύνολο από συναρτήσεις που επιτελούν βασικές διεργασίες. Η γνώση των συναρτήσεων αυτών αποτελεί βασική προϋπόθεση για την ανάπτυξη σύνθετων προγραμμάτων, όπου η επαναχρησιμοποίηση έτοιμων ελεγμένων τμημάτων κώδικα μειώνει σημαντικά το χρόνο ανάπτυξης, αυξάνοντας ταυτόχρονα την αξιοπιστία. Είναι σημαντικό στη φάση αυτή, να αποκαταστήσετε μια πρόσβαση σε τεκμηρίωση της βασικής βιβλιοθήκης και σιγά–σιγά να εξοικειώνεστε με τις συναρτήσεις που αυτή περιέχει. Ξεκινήστε από τις βασικές κατηγορίες διαχείρισης αλφαριθμητικών, μαθηματικές, ταξινόμησης χαρακτήρων και μετατροπές χαρακτήρων, εισόδου εξόδου και συνεχίστε σε επόμενα κεφάλαια και όταν απαιτηθεί με τις κατηγορίες μετατροπής δεδομένων, αναζήτησης και ταξινόμησης, διαχείρισης αρχείων, διαχείρισης μνήμης.



Σχήμα 6.1.

Υπολογισμός της τιμής της έκφρασης `max(max(15, 13), 17)`*. Η διακεκομμένη γραμμή παριστά απόδοση τιμών στις παραμέτρους και την επιστρεφόμενη τιμή, η δε συνεχής τη ροή του ελέγχου.*

Δραστηριότητα 6.2

Γράψτε ένα πρόγραμμα στο οποίο θα κάνετε χρήση των συναρτήσεων `strcpy()`, `strcat()`, `strlen()`, `strcmp()`, και `strchr()` της βασικής βιβλιοθήκης. Τη δική μας έκδοση θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 6.3

Γράψτε ένα πρόγραμμα στο οποίο θα κάνετε χρήση των συναρτήσεων `isalnum()`, `isalpha()`, `isdigit()`, `islower()`, `isspace()`, `tolower()`, και `toupper()`. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 6.4

Γράψτε ένα πρόγραμμα στο οποίο θα κάνετε χρήση των συναρτήσεων `gets()`, `puts()`, `getchar()`, και `putchar()`. Τη δική μας έκδοση θα βρείτε στο τέλος του βιβλίου.

Κανόνες βελτίωσης αναγνωσιμότητας προγράμματος

Οι παρακάτω οδηγίες θα σας βοηθήσουν να βελτιώσετε την αναγνωσιμότητα των προγραμμάτων σας:

1. Οργανώστε τον κώδικά σας σύμφωνα με το σχήμα
   εντολές προεπεξεργαστή (πρώτα `#include` μετά `#define`)
   πρωτότυπα συναρτήσεων
   δηλώσεις γενικών μεταβλητών
   συνάρτηση `main`
   ορισμοί λοιπών συναρτήσεων
2. Συνηθίστε να αφήνετε 2–3 κενές γραμμές μεταξύ των παραπάνω ομάδων προτάσεων, αυξάνει σε μεγάλο βαθμό την αναγνωσιμότητα του πηγαίου κώδικα.
3. Χρησιμοποιήστε εκφραστικά ονόματα μεταβλητών και συναρτήσεων όπως `max_velocity`, `max_temp`, `word_len`, `display_error(...)`, `isdigit(...)`, `read_ten_chars(...)`.
4. Παρεμβάλετε σχόλια σε όποια σημεία του προγράμματος κρίνετε σκόπιμο ότι αυτά θα βελτιώσουν την αναγνωσιμότητα του κώδικα.
5. Εφαρμόζετε τον κανόνα της μιας πρότασης ανά γραμμή.
6. Βάλτε όλες τις δηλώσεις μεταβλητών σε μια θέση.
7. Αφιερώστε πριν τη συγγραφή του πηγαίου κώδικα αρκετό χρόνο για το σχεδιασμό του προγράμματός σας.

Σύνοψη

Διαδικασία και συνάρτηση αποτελούν τις δύο βασικές γλωσσικές κατασκευές που χρησιμοποιούνται για αναπαράσταση της διεργασίας. Υποστηρίζουν την εφαρμογή της αφαιρετικότητας, γιατί επιτρέπουν στον προγραμματιστή να εστιάσει την προσοχή του στη διεργασία που εκτελείται στο σημείο κλήσης τους και όχι στον τρόπο με τον οποίο αυτή επιτελείται. Αυτό επιτυγχάνεται διακρίνοντας δύο τμήματα, τη διεπαφή και την υλοποίηση. Η διεπαφή που προσδιορίζει πλήρως τη συνάρτηση, αποτελείται από το όνομά της που πρέπει να περιγράφει το έργο (διεργασία) που η συνάρτηση εκτελεί, προαιρετικά δε από ένα σύνολο από παραμέτρους και μια επιστρεφόμενη τιμή. Η διεπαφή αυτή, γνωστή στη C σαν πρωτότυπο της συνάρτησης, επιτρέπει στον μεταγλωττιστή να διασφαλίσει ότι κάθε κλήση της συνάρτησης είναι σύμφωνη με τον ορισμό της, προφυλάσσοντας τον προγραμματιστή από δύσκολα στην ανεύρεση λάθη.

Η βασική βιβλιοθήκη, ένα σύνολο από ελεγμένες γενικού σκοπού συναρτήσεις έτοιμες για χρήση, αποτελεί απαραίτητο εφόδιο στα χέρια του προγραμματιστή για τη γρήγορη ανάπτυξη αξιόπιστου κώδικα.

Βιβλιογραφία

[Darnell 1991]

Darnell P., Margolis P. «*C: A Software Engineering Approach*», Springer–Verlag, New York.

[Rojiani ’96]

Rojiani K.B., «*Programming in C with numerical methods for Engineers*», Prentice–Hall.

[Sethi ’97]

Sethi Ravi, «*Programming Languages: Concepts and Constructs*» 2nd Edition, Addison Wesley 1996. Reprinted with corrections April ’97.

ΚΕΦΑΛΑΙΟ 7

Προτάσεις Ελέγχου Ροής

Σκοπός

Σκοπός του κεφαλαίου είναι να εισάγει την έννοια της ροής ελέγχου σε ένα πρόγραμμα, δίνοντας ταυτόχρονα τις βασικές κατηγορίες προτάσεων που επιτρέπουν στον προγραμματιστή να διαμορφώσει τη ροή εκτέλεσης ανάλογα με τις απαιτήσεις της κάθε εφαρμογής.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *περιγράψετε τις βασικές κατηγορίες προτάσεων διαμόρφωσης της ροής ελέγχου σε ένα πρόγραμμα,*
- *αναγνωρίσετε ποια από τις κατηγορίες προτάσεων ελέγχου ροής θα πρέπει να χρησιμοποιήσετε σε δεδομένη περίπτωση,*
- *δώσετε τρεις τουλάχιστον προτάσεις για επανάληψη,*
- *δώσετε δυο τουλάχιστον προτάσεις για υπό συνθήκη διακλάδωση,*
- *γράψετε τη σύνταξη των προτάσεων ελέγχου ροής της C,*
- *εξηγήσετε τη διαφορά μεταξύ βρόχου επανάληψης συνθήκης–εισόδου και αντίστοιχου συνθήκης–εξόδου,*
- *να διαμορφώσετε τη ροή ελέγχου του προγράμματός σας ώστε να ανταποκρίνεται στις απαιτήσεις που έχετε από αυτό.*

Έννοιες κλειδιά

- *δομημένος προγραμματισμός (structured programming)*
- *πρόταση επανάληψης*
- *πρόταση διακλάδωσης*
- *διακλάδωση υπό συνθήκη*
- *διακλάδωση χωρίς συνθήκη*
- *επανάληψη υπό συνθήκη (conditional loop)*
- *απαριθούμενη επανάληψη (counting loop)*
- *τελεστής κόμμα*
- *ένθετοι βρόχοι*
- `if`
- `if else`
- `switch case`
- `for`
- `while`
- `do while`
- `break`
- `continue`

Εισαγωγικές Παρατηρήσεις

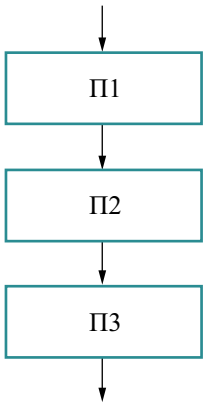
Ένα πρόγραμμα στην προστακτική μορφή προγραμματισμού έχει σαν βασική μονάδα την ενέργεια η οποία αναπαρίσταται από την πρόταση. Μια πρόταση μπορεί να μεταβάλει τις τιμές των δεδομένων, τα οποία αναπαρίστανται από μεταβλητές. Επειδή η σειρά αυτών των μεταβολών έχει άμεση επίδραση στην πορεία εξεύρεσης της λύσης, ένα από τα σημαντικά έργα του προγραμματιστή στην προστακτική μορφή προγραμματισμού είναι ο καθορισμός της ροής ελέγχου του προγράμματος, δηλαδή ο καθορισμός της σειράς εκτέλεσης των εντολών ή των προτάσεων από τις οποίες απαρτίζεται το πρόγραμμα.

Το κεφάλαιο αυτό είναι οργανωμένο σε δύο ενότητες. Στην πρώτη, εξετάζουμε γενικά και ανεξαρτήτως γλώσσας προγραμματισμού, το θέμα της διαμόρφωσης της ροής ελέγχου του προγράμματος. Περιγράφουμε τις βασικές κατηγορίες προτάσεων, που θα σας επιτρέψουν να διαμορφώσετε ανάλογα με την εφαρμογή, τη ροή ελέγχου του αντίστοιχου προγράμματός σας, διαφοροποιώντας την από την κλασσική ακολουθιακή εκτέλεση. Στη δεύτερη ενότητα, παρουσιάζουμε διεξοδικά τις αντίστοιχες προτάσεις ροής ελέγχου της C, περιγράφοντας για κάθε μια τον τρόπο χρήσης της και δίνοντας αντιπροσωπευτικά παραδείγματα.

7.1 Η διαμόρφωση της ροής ελέγχου προγράμματος

7.1.1 Δομημένος προγραμματισμός

Ο πλέον συνηθισμένος τρόπος εκτέλεσης που συναντάμε σε κάθε προστακτική γλώσσα είναι ο ακολουθιακός. Δύο ή περισσότερες προτάσεις διατεταγμένες η μία μετά την άλλη εκτελούνται διαδοχικά, όπως φαίνεται στο σχήμα 7.1. Έτσι, η ακολουθία των προτάσεων Π1; Π2; Π3; έχει σαν αποτέλεσμα την εκτέλεση της πρότασης Π1, μετά της Π2 και τέλος της Π3.



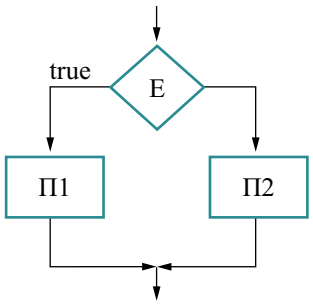
Σχήμα 7.1.
Ακολουθιακή εκτέλεση προτάσεων.

Για να επιτευχθεί οποιαδήποτε διαφοροποίηση από την ακολουθιακή εκτέλεση χρησιμοποιούνται ειδικές κατασκευές. Ορισμένες από τις κατασκευές που χρησιμοποιούνται για να επιτευχθεί η επιθυμητή ροή ελέγχου, διασφαλίζουν ταυτόχρονα τη δόμηση του προγράμματος με κύριο στόχο αυτόν που εκφράζεται από την ακόλουθη πρόταση. **«Η δομή του πηγαίου κώδικα θα πρέπει να μας βοηθά να κατανοήσουμε τι κάνει το πρόγραμμα».** Μια γλώσσα που έχει κατασκευές ροής ελέγχου που υποστηρίζουν αυτή την αρχή, λέγεται *γλώσσα δομημένου προγραμματισμού*, ο δε προγραμματισμός με τέτοιες κατασκευές, *δομημένος προγραμματισμός* (structured programming). Ο αναγνώστης δομημένου πηγαίου κώδικα μπορεί εύκολα να κατανοήσει τι συμβαίνει όταν το πρόγραμμα εκτελείται.

Οι κατασκευές που ευνοούν την ανάπτυξη δομημένου κώδικα ορίζονται σαν *μιας εισόδου – μιας εξόδου* (single–entry/single–exit). Η C αλλά και η Pascal διαθέτουν γλωσσικές κατασκευές για την υποστήριξη της ακολουθιακής εκτέλεσης, της επιλογής και της επανάληψης που ανήκουν σε αυτή την κατηγορία. Η μόνη κατασκευή των γλωσσών αυτών που επιτρέπει παραβίαση του κανόνα μιας εισόδου–εξόδου είναι η κατασκευή `goto`, η οποία όμως στην πράξη χρησιμοποιείται ελάχιστα έως καθόλου.

Η διαμόρφωση της ροής ελέγχου, στο δομημένο προγραμματισμό, επιτυγχάνεται με την κατάλληλη χρήση προτάσεων των δύο βασικών κατηγοριών της *επανάληψης* (looping) και της *υπό συνθήκη διακλάδωσης* (conditional branching). Σύμφωνα με την κατασκευή επανάληψης, μία πρόταση ή ένα σύνολο προτάσεων εκτελείται επανειλημμένα, ενώ σύμφωνα με την κατασκευή της υπό συνθήκη διακλάδωσης, διαφορετικές προτάσεις ή ομάδες προτάσεων εκτελούνται ανάλογα με την τιμή μιας έκφρασης.

Η επίδραση της δόμησης στην *αποδοτικότητα* (efficiency) του προγράμματος είναι μηδαμινή. Προσεκτικά σχεδιασμένος δομημένος κώδικας είναι εξί-



Σχήμα 7.2.
Υπό συνθήκη εκτέλεση δύο προτάσεων.

σου αποδοτικός όσο ο αντίστοιχος μη δομημένος. Επιπλέον, είναι ευκολότερο να βελτιώσεις την αποδοτικότητα ενός δομημένου πηγαίου κώδικα από ό,τι ενός μη δομημένου.

7.1.2 Προτάσεις διακλάδωσης υπό συνθήκης

Μια πρόταση διακλάδωσης υπό συνθήκη, περιέχει έναν αριθμό υπο–προτάσεων, από τις οποίες επιλέγεται και εκτελείται μόνο μία. Η πρόταση `if`, την οποία συναντάμε σε όλες τις προστακτικές γλώσσες είναι η πλέον γνωστή πρόταση αυτής της κατηγορίας και έχει την μορφή:

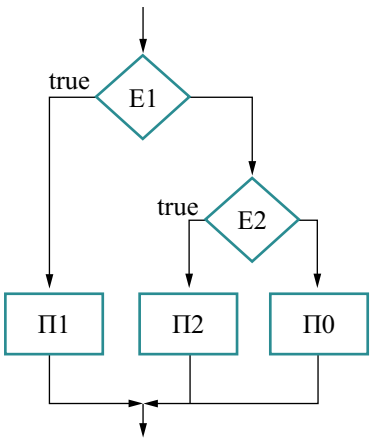
if Ε **then** Π1 **else** Π2

Είναι προφανές, όπως φαίνεται και από το σχήμα 7.2 που αναπαριστά την κατασκευή αυτή, ότι η πρόταση είναι μιας εισόδου–μιας εξόδου. Ο έλεγχος εισέρχεται από το σημείο στην κορυφή, υπολογίζεται η τιμή της λογικής έκφρασης Ε, και αν είναι αληθής επιλέγεται και εκτελείται η πρόταση Π1 διαφορετικά η Π2. Σε κάθε περίπτωση, ο έλεγχος μεταφέρεται στο ένα και μοναδικό σημείο εξόδου στο κάτω μέρος του διαγράμματος.

Σε μια πιο σύνθετη μορφή της η πρόταση `if` επιτρέπει επιλογή από μεγαλύτερο (συνήθως απεριόριστο) αριθμό προτάσεων, με την ένθεση διαδοχικών προτάσεων συνθήκης. Στην περίπτωση αυτή, το σχήμα που βελτιώνει την αναγνωσιμότητα του κώδικα έχει την παρακάτω μορφή:

if Ε1 **then** Π1
else if Ε2 **then** Π2
else if Ε3 **then** Π3
:
else if Εn **then** Πn
else Π0

Οι λογικές εκφράσεις υπολογίζονται σειριακά και η πρώτη που θα δώσει αληθή τιμή οδηγεί στην εκτέλεση της αντίστοιχης πρότασης. Αν, για παράδειγμα, Ε1 και Ε2 είναι ψευδείς και η Ε3 δώσει αληθή τιμή, τότε εκτελείται η Π3. Αν καμμία από τις λογικές εκφράσεις Ε δεν δώσει αληθή τιμή εκτελείται η πρόταση Π0.



Σχήμα 7.3.
Επιλεκτική εκτέλεση πολλών προτάσεων με ένθεση.

Είναι προφανές, όπως φαίνεται και από το flow–chart του σχήματος 7.3, που αναπαριστά ένθεση μιας πρότασης με συνθήκη Ε2, ότι και η κατασκευή αυτή ικανοποιεί τον κανόνα μιας εισόδου–μιας εξόδου.

Μια απλοποιημένη παραλλαγή της πρότασης συνθήκης if έχει τη μορφή

if Ε **then** Π

Στην πρόταση αυτή υπολογίζεται η λογική έκφραση *Ε* και, αν η τιμή της είναι αληθής, εκτελείται η πρόταση *Π*, αλλιώς ο έλεγχος μεταφέρεται στην πρόταση που είναι μετά την `if`.

Χαρακτηριστικό της πρότασης `if` είναι ότι σε κάθε περίπτωση μπορεί να προβλεφθεί ποια πρόταση θα εκτελεστεί, είναι όπως λέμε ντετερμινιστική. Μη ντετερμινιστικές προτάσεις συνθήκης συναντώνται σε γλώσσες προγραμματισμού που υποστηρίζουν ταυτόχρονο προγραμματισμό (concurrent programming), όπως η Ada [Watt '90].

Η πρόταση `if` με τις παραλλαγές της βασίζει την επιλογή της σε λογική έκφραση. Παρόλα αυτά, η C (όπως και η Pascal και η Ada) επιτρέπει να χρησιμοποιηθούν σαν εκφράσεις επιλογής, εκφράσεις τύπου απαριθμητικού, χαρακτήρα ή ακέραιου[1].

Η `switch` (δες υποενότητα 7.2.2), μια εξίσου χρήσιμη πρόταση υπό συνθήκη διακλάδωσης, επιτρέπει στον προγραμματιστή να επιλέξει μία από ένα σύνολο από αμοιβαία αποκλειόμενες επιλογές. Η εντολή, γνωστή γενικότερα σαν `case`, έχει την αρχή της από την ALGOL. Στην Pascal η πρόταση εμφανίζεται επίσης σαν `case`, ενώ στην C είναι γνωστή σαν `switch case`.

Δραστηριότητα 7.1

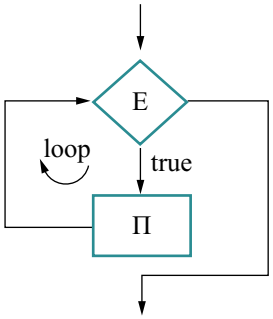
Περιγράψτε με ψευδοκώδικα τη διεργασία που πρέπει να εκτελέσει ο υπολογιστής για να διαπιστώσει αν δεδομένο έτος είναι δίσεκτο. Χρησιμοποιήστε την κατασκευή `if then else` και τον τελεστή %. Τα σχόλιά μας θα βρείτε στο τέλος του βιβλίου.

7.1.3 Προτάσεις επανάληψης

Οι προτάσεις επανάληψης (iterative ή loop statements) διακρίνονται σε δύο κατηγορίες ανάλογα με το αν γνωρίζουμε τον αριθμό των επαναλήψεων ή όχι.

Μια πρόταση ή ένα σύνολο προτάσεων μπορεί να εκτελείται όσο μια συνθήκη είναι αληθής (conditional loop) ή για προκαθορισμένο αριθμό επαναλήψεων (counting loop).

[1] Αντίθετα η ML είναι η γλώσσα που δεν βάζει σχεδόν κανένα περιορισμό στον τύπο της έκφρασης [Watt 90].



Σχήμα 7.4.

Επανάληψη με συνθήκη εισόδου.

Μια πρόταση επανάληψης αποτελείται συνήθως από δύο μέρη. Το ένα είναι μία πρόταση, απλή ή σύνθετη, που ονομάζεται σώμα της επανάληψης (loop body). Το δεύτερο είναι μια έκφραση που προσδιορίζει πότε η επανάληψη θα τερματιστεί. Ορισμένες προτάσεις περιλαμβάνουν επιπλέον, όπως θα δούμε στη συνέχεια, και μια έκφραση αρχικοποίησης και μια έκφραση ανανέωσης.

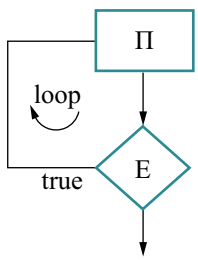
Σε μια αορίστων επαναλήψεων πρόταση (indefinite iterations), όπως διαφορετικά ονομάζεται η υπό συνθήκη επανάληψη, ο αριθμός των επαναλήψεων δεν είναι γνωστός τη στιγμή που ο έλεγχος φτάνει στην πρόταση. Κλασσικό παράδειγμα αυτής της κατηγορίας είναι η πρόταση `while` που στην Pascal έχει την μορφή

while E **do** Π

Η πρόταση *Π* είναι το σώμα της πρότασης επανάληψης και επαναλαμβάνεται όσο η έκφραση *E* είναι αληθής, όπως φαίνεται στο σχήμα 7.4. Όταν η τιμή της έκφρασης γίνει ψευδής ο έλεγχος μεταφέρεται στην επόμενη πρόταση. Η πρόταση αναφέρεται και σαν *συνθήκης–εισόδου*. Ο έλεγχος μπορεί να φύγει από την πρόταση χωρίς το σώμα της να εκτελεστεί έστω για μία φορά, σε αντίθεση με την αντίστοιχη *συνθήκης–εξόδου,* η οποία διασφαλίζει την εκτέλεση του σώματος της πρότασης τουλάχιστον για μία φορά. Η μορφή της πρότασης επανάληψης με συνθήκη εξόδου για την C είναι

do Π **while** E

Όπως φαίνεται στο σχήμα 7.5, η πρόταση *Π* εκτελείται πριν τον έλεγχο της συνθήκης *E*, και αυτό διασφαλίζει την για μία τουλάχιστον φορά εκτέλεσή της. Ο έλεγχος φεύγει από το βρόχο όταν η έκφραση *E* δώσει ψευδή τιμή.



Σχήμα 7.5.

Επανάληψη με συνθήκη εξόδου.

Η Pascal διαθέτει μια παραλλαγή με τη μορφή

repeat Π **until** E

όπου η έξοδος από το loop γίνεται όταν η έκφραση E δώσει αληθή τιμή.

Τη σημασία της πρότασης `repeat` μπορούμε να αποδώσουμε με την πρόταση `while` ως:

Π; **while** not E **do** Π

Αντίστοιχα, η σημασία της `while` μπορεί να αποδοθεί με την `repeat`, όπως παρακάτω

if E **then repeat** Π **until not** E

Σε μια πρόταση προκαθορισμένου αριθμού επαναλήψεων, ο αριθμός των επαναλήψεων ελέγχεται συνήθως από μια *μεταβλητή ελέγχου* (control variable). Η μεταβλητή συνήθως παίρνει μια αρχική τιμή, σε κάθε επανάληψη παίρνει την επόμενη από μια προκαθορισμένη ακολουθία τιμών (control sequence) και το σώμα επαναλαμβάνεται έως ότου η μεταβλητή ελέγχου φτάσει στο όριο της ακολουθίας τιμών. Η πρόταση της Pascal που υλοποιεί αυτή τη λογική έχει τη γενική μορφή

for <όνομα> **:=** Ε1 **to** Ε2 **do** Π

Η μεταβλητή της οποίας το όνομα προσδιορίζεται, παίρνει σαν αρχική τιμή την τιμή της έκφρασης *Ε1* και, στη συνέχεια, επαναλαμβάνεται η εκτέλεση της πρότασης *Π*, καθόσον η μεταβλητή έχει τιμή μικρότερη της τιμής της έκφρασης *Ε2*. Μια αναλυτική αναφορά μπορείτε να βρείτε στο [Sethi '97]. Η C διαθέτει μια πιο ισχυρή και ευέλικτη μορφή που θα δούμε αναλυτικά στη συνέχεια του κεφαλαίου.

7.1.4 Προτάσεις διακλάδωσης χωρίς συνθήκη

ΔΙΑΧΕΙΡΙΣΗ ΕΙΔΙΚΩΝ ΠΕΡΙΠΤΩΣΕΩΝ ΣΕ ΠΡΟΤΑΣΕΙΣ ΕΠΑΝΑΛΗΨΗΣ

Οι προτάσεις `break` και `continue` διευκολύνουν τη διαχείριση ειδικών περιπτώσεων μέσα σε βρόχους επανάληψης. Η εντολή `break` μεταφέρει τον έλεγχο έξω από τον βρόχο στον οποίο εμπεριέχεται και, πιο συγκεκριμένα, στην πρόταση που ακολουθεί το βρόχο. Χρησιμοποιείται, επίσης, και στην πρόταση switch, όπως θα δούμε στη συνέχεια, για να μεταφέρει τον έλεγχο στην πρόταση που ακολουθεί την `switch`. Ο ψεοδοκώδικας που ακολουθεί δίνει ένα παράδειγμα χρήσης της `break`.

```
while (έκφραση) {
   if (ειδική περίπτωση) {
        προτάσεις επεξεργασίας ειδικής περίπτωσης
        break;
        }
   προτάσεις επεξεργασίας κανονικών περιπτώσεων
   }
```

Εκτός από την `switch`, όπου η `break` είναι αναντικατάστατη, στις άλλες περιπτώσεις υπάρχει πάντα τρόπος να γραφεί κώδικας χωρίς τη χρήση της. Πολλές `break` καταστρέφουν τη δόμηση του κώδικα και δημιουργούν προβλήματα στην παρακολούθηση της ροής ελέγχου.

Η εντολή `continue` μεταφέρει τον έλεγχο στην αρχή του βρόχου. Χρησιμοποιείται, συνήθως, όταν θέλουμε να μεταφέρουμε τον έλεγχο στην επόμενη επανάληψη του βρόχου παραλείποντας την εκτέλεση του υπόλοιπου τμήματος του σώματος του βρόχου. Ο ψευδοκώδικας που δώσαμε παραπάνω με την break μπορεί να δοθεί και με χρήση της `continue`, όπως παρακάτω:

```
while (έκφραση) {
   if (κανονική περίπτωση) {
        προτάσεις επεξεργασίας κανονικής περίπτωσης
        continue;
        }
   προτάσεις επεξεργασίας ειδικών περιπτώσεων
   }
```

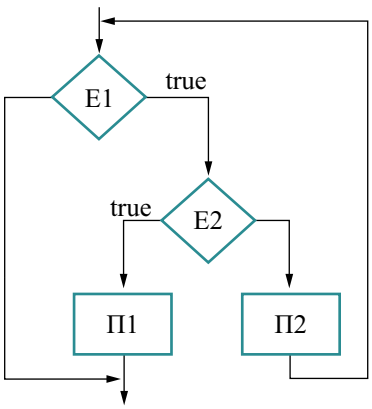
Ο ίδιος κώδικας θα μπορούσε να γραφεί χωρίς την χρήση της `continue` με την παρακάτω μορφή:

```
while (έκφραση) {
   if (κανονική περίπτωση)
        προτάσεις επεξεργασίας κανονικής περίπτωσης
   else
        προτάσεις επεξεργασίας ειδικών περιπτώσεων
   }
```

Σαν πλεονέκτημα των εκδόσεων με τις `break` και `continue` μπορεί να αναφερθεί το μικρότερο βάθος στηλοθέτισης (indentation) των «προτάσεων επεξεργασίας των ειδικών περιπτώσεων» που σε περιπτώσεις πολλών indentations κρατά τον κώδικα κοντά στο αριστερό περιθώριο της οθόνης.

Σχήμα 7.6.

Η break σε πρόταση επανάληψης



Στο σχήμα 7.6 φαίνεται ότι η πρόταση `while` εξακολουθεί να είναι δομή μιας εισόδου–μιας εξόδου και με τη χρήση της `break`. Ο αντίστοιχος ψευδοκώδικας είναι

```
while(E1) {
   if(E2) {
        Π1
        break;
        }
   Π2
   }
```

ΜΕΤΑΦΟΡΑ ΕΛΕΓΧΟΥ ΑΠΟ ΣΥΝΑΡΤΗΣΗ ΣΤΗΝ ΚΑΛΟΥΣΑ ΣΥΝΑΡΤΗΣΗ.

Στην κατηγορία αυτή ανήκει η εντολή `return`, η οποία όπως είδαμε, μεταφέρει τον έλεγχο πίσω στην καλούσα συνάρτηση επιστρέφοντας ταυτόχρονα και την τιμή της έκφρασης που ακολουθεί την `return`. Να υπενθυμίσουμε ότι, εφόσον το επιβάλει η λογική του αλγόριθμου τον οποίο υλοποιεί η συνάρτηση, μπορεί να υπάρχουν περισσότερες της μιας προτάσεις return σε μια συνάρτηση.

ΡΗΤΗ ΔΙΑΚΛΑΔΩΣΗ

Η πρόταση `goto` **<ετικέτα>** μεταφέρει τον έλεγχο στην πρόταση που σημειώνεται με την ετικέτα σαν **<ετικέτα> : πρόταση**

Η C κατακρίνεται από πολλούς για την παροχή της `goto` εντολής. Υπάρχουν όμως περιπτώσεις, όπως αυτή της εξόδου από πολύ βαθιά ενσωματωμένη δομή, που μια προσεκτική χρήση της δίνει πιο συμπαγή κώδικα. Δύο παραδείγματα αποδεκτής χρήσης της μπορείτε να δείτε στο [Kernighan '88] αλλά και μια εξίσου καλή αιτιολόγηση του λόγου ύπαρξής της δίνεται στο [Sethi '97]. Η συμβουλή μας; Μην χρησιμοποιείται την `goto` πριν γίνετε πολύ καλοί C προγραμματιστές. Μετά; Θα διαπιστώσετε ότι δεν είναι απαραίτητη. Είναι πάντα δυνατό να γράφεις κώδικα χωρίς `goto`.

7.2 Προτάσεις ελέγχου ροής στη C

Εισαγωγικές Παρατηρήσεις

Η C παρέχει ένα πολύ ευέλικτο σύνολο προτάσεων για τον έλεγχο της ροής εκτέλεσης του προγράμματος. Οι προτάσεις αυτές σε συνδυασμό με το πλούσιο σύνολο τελεστών δίνουν τη δυνατότητα στον προγραμματιστή να δημιουργεί αποδοτικό και ταυτόχρονα ευανάγνωστο κώδικα. Παρόλα αυτά, η παραγωγή δομημένου κώδικα εξαρτάται άμεσα από τον προγραμματιστή και τον τρόπο που χρησιμοποιεί τα δομικά στοιχεία της γλώσσας. Η ενότητα αυτή παρουσιάζει αφενός μεν τις προτάσεις ροής ελέγχου, αφετέρου δε τον τρόπο που αυτές πρέπει να χρησιμοποιούνται με στόχο πάντα τη δημιουργία δομημένου κώδικα. Οι προτάσεις διακρίνονται στις παρακάτω κατηγορίες:

- *Προτάσεις διακλάδωσης υπό συνθήκη:* `if–else, switch case`
- *Προτάσεις επανάληψης:* `while, do while, for`
- *Προτάσεις διακλάδωσης χωρίς συνθήκη:* `break, continue, goto`

7.2.1 Πρόταση if

Η πρόταση `if` είναι σύμφωνη με τον γενικό κανόνα που δώσαμε, και στην απλούστερη μορφή της δίνεται με την παρακάτω σύνταξη:

`if` (έκφραση1)
   πρόταση1;

Η έκφραση, που τις περισσότερες φορές είναι συσχετιστική, υπολογίζεται και αν η τιμή της είναι αληθής (κάθε τιμή διάφορη του 0 για την C, όπως θα θυμόσαστε είναι αληθής) εκτελείται η πρόταση *πρόταση1*, αλλιώς ο έλεγχος μεταφέρεται στην επόμενη πρόταση. Η σύνταξη της πρότασης if είναι πολύ κοντά στην ομιλούμενη, γεγονός που μπορεί να διαπιστωθεί από την παραπλεύρως πρόταση που αποδίδει σε C την πρόταση της ομιλούμενης «εάν το φανάρι είναι κόκκινο, σταμάτα, αλλιώς προχώρα».

```
if (light == red)
   stop();
else
   go();
```

Η *πρόταση1* του `if` μπορεί να είναι απλή αλλά και σύνθετη. Στη δεύτερη αυτή περίπτωση, η μορφή τής `if` έχει όπως παρακάτω:

```
if (έκφραση) {
   πρόταση1;
   …
   πρότασηN;
   }
```

Το ίδιο ισχύει και για την πρόταση που ακολουθεί το `else`.

Ευρεία χρήση στην πρόταση `if`, βρίσκουν οι συσχετιστικοί τελεστές δημιουργώντας εκφράσεις οι παρακάτω:

Α) `if( (num >10) && (num < 14))` Β) `if((ch == '\n') || (ch =='\t'))`
Γ) `if(!found)` Δ) `if((ch>'a') && (ch<'z'))`

Παράδειγμα 7.1

Να γραφεί πρόγραμμα που να ζητά από τον χρήστη ένα μη αρνητικό αριθμό και να τυπώνει την τετραγωνική του ρίζα.

Η περιγραφή σε δομημένα Ελληνικά, της διεργασίας που πρέπει να ακολουθή-

σει ο υπολογιστής και ο αντίστοιχος κώδικας σε C έχουν την παρακάτω μορφή:

```
Πάρε τον αριθμό
εάν ο αριθμός είναι αρνητικός
    τύπωσε μήνυμα λάθους
αλλιώς
    τύπωσε την τετραγωνική ρίζα του
    αριθμού
τερμάτισε
```

```
#include <stdio.h>
#include <math.h> /* βιβλιοθήκη μαθηματικών συναρτήσεων */
main()
{
double num;
printf("Δώσε ένα θετικό αριθμό:);
scanf("%lf», &num);
if(num<0)
   printf("Λάθος είσοδος: Αριθμός αρνητικός\n");
else
   printf("Η τετραγωνική ρίζα του %lf είναι
%f\n", sqrt(num));
exit(0);
}
```

Προσέξτε την τοποθέτηση των προτάσεων που ακολουθούν το `if` και `else` στην επόμενη θέση του στηλογνώμονα. Η σύμβαση αυτή βοηθά πολύ στη βελτίωση της αναγνωσιμότητας του κώδικα και θα πρέπει να την τηρείτε ευλαβικά.

Παράδειγμα 7.2

Δώστε τη δήλωση και τον ορισμό συνάρτησης για τον υπολογισμό του μέγιστου μεταξύ δύο ακέραιων αριθμών.

Η συνάρτηση θα δέχεται δύο ακέραια ορίσματα και θα επιστρέφει το μεγαλύτερο. Η δήλωση της συνάρτησης θα έχει τη μορφή:

```
int max(int a, int b);
```

Ο ορισμός της μπορεί να έχει μια από τις παρακάτω μορφές:

```
int max(int a, int b)
{
int max;
if (a>b)
     max = a;
else
     max = b;
return(max)
}
```

```
int max(int a, int b)
{
if (a>b)
     return a;
else
     return b;
}
```

```
int max(int a, int b)
{
return (a>b) ? a : b ;
}
```

α) πρώτη έκδοση της max β) δεύτερη έκδοση της max γ) τρίτη έκδοση της max

Στη δεύτερη έκδοση για να αποφύγουμε τη χρήση της τοπικής μεταβλητής `max`, χρησιμοποιούμε τη `return` δύο φορές, με διαφορετική, βέβαια, σε κάθε περίπτωση, επιστρεφόμενη τιμή.

Ενδιαφέρον παρουσιάζει η τρίτη έκδοση, που αποτελεί χαρακτηριστικό παράδειγμα γραφής συμπαγούς κώδικα. Παρατηρήστε την αντιστοίχηση του υποθετικού τριαδικού τελεστή `? :` με την πρόταση `if else`. Είναι σαφές πως ο τελεστής μάς δίνει τη δυνατότητα να εισάγουμε σε εκφράσεις τη λογική που εκφράζει η πρόταση `if else`.

Δραστηριότητα 7.2

Δώστε τη δήλωση και τον ορισμό συνάρτησης για τον υπολογισμό του μέγιστου μεταξύ τριών πραγματικών αριθμών. Μπορεί να χρησιμοποιηθεί ο υποθετικός τελεστής (`? :`); Τα σχόλιά μας θα βρείτε στο τέλος του βιβλίου.

7.2.2 Η πρόταση `switch`

Η πρόταση `switch` επιτρέπει τον προσδιορισμό απεριόριστου αριθμού διαδρομών ανάλογα με την τιμή μιας έκφρασης. Η γενική μορφή της πρότασης switch δίνεται στο σχήμα 7.7, εκτελείται δε όπως περιγράφεται στη συνέχεια. Υπολογίζεται η *έκφραση*, και η τιμή της συγκρίνεται διαδοχικά με τις σταθερές εκφράσεις (*σταθ–εκφρ1, σταθ–εκφρ2, σταθ–εκφρ3,...).* Ο έλεγχος μεταφέρεται κάτω από την σταθερά έκφραση με την οποία η τιμή της έκφρασης ισούται, δηλαδή σε μία εκ των *πρόταση1, πρόταση2, ...* Αν δεν ισούται με καμία από τις σταθερές εκφράσεις, ο έλεγχος μεταφέρεται στην πρόταση που ακολουθεί την ετικέτα *`default`*, εάν βέβαια αυτή υπάρχει, αλλιώς στην πρόταση μετά την ολοκλήρωση του σώματος της `switch`. Ένα σημαντικό στοιχείο είναι ότι η ροή του προγράμματος συνεχίζει από την επιλεχθείσα `case` ετικέτα μέχρι το τέλος της πρότασης `switch` ή μέχρι να συναντηθεί μια πρόταση άμεσης μεταφοράς ελέγχου. Αυτό σημαίνει πως το σύστημα εκτελεί τις προτάσεις κάτω από την επιλεχθείσα ετικέτα έως ότου συναντήσει πρόταση `break`, `goto` ή `return`. Η πρόταση ελέγχου `break`, αν και πολλές φορές δίνεται στη γενική μορφή της `switch`, είναι προαιρετική και η εκτέλεσή της μεταφέρει τον έλεγχο έξω από την πρόταση `switch`. Έτσι, αν για

παράδειγμα λείπουν οι προτάσεις `break` από μια `switch`, η εκτέλεση των προτάσεων που ακολουθούν την επιλεχθείσα ετικέτα, θα ακολουθηθεί από την εκτέλεση και των προτάσεων των επόμενων `case` ετικετών. Παρόλο, που η `break` είναι προαιρετική, στην πράξη τη συναντάμε σχεδόν πάντα, ακόμη και μετά τις προτάσεις της τελευταίας ετικέτας. Στην τελευταία αυτή περίπτωση, μας προστατεύει από το δύσκολο στην ανεύρεση σφάλμα που θα προκύψει από μελλοντική προσθήκη μιας νέας ετικέτας με ταυτόχρονη παράλειψη προσθήκης πριν από αυτή της πρότασης break.

Η λειτουργία της `switch` διέπεται από ένα σύνολο κανόνων, οι σημαντικότεροι από τους οποίους είναι οι εξής:

- Κάθε `case` πρέπει να έχει μια `int` ή `char` σταθερά ή μια σταθερά έκφραση.
- Δύο `case` δεν μπορούν να έχουν εκφράσεις με την ίδια τιμή.
- Οι προτάσεις κάτω από την ετικέτα `default` εκτελούνται όταν καμία από τις `case` ετικέτες δεν ικανοποιείται.
- `Case` και `default` μπορούν να τοποθετηθούν με οποιαδήποτε σειρά. Αυτό σημαίνει πως η default δεν είναι απαραίτητα η τελευταία ετικέτα. Παρόλα αυτά, εκτελείται όταν δεν επιλεγεί καμία `case` προηγούμενη ή επόμενη.
- Η break μετά τη τελευταία ετικέτα αποτελεί καλή τακτική αν και δεν είναι απαραίτητη.

***switch** (έκφραση) {*
***case** σταθ–εκφρ1 :*
πρόταση1
***case** σταθ–εκφρ2 :*
πρόταση2
***case** σταθ–εκφρ3 :*
πρόταση3
***default** :*
πρόταση
}

Σχήμα 7.7
Πρόταση πολλαπλών επιλογών

Παράδειγμα 7.3

Να δημιουργηθεί πρόγραμμα το οποίο να δίνει τη δυνατότητα στο χρήστη να εισάγει δύο αριθμούς και, στη συνέχεια, να εκτελεί πάνω σε αυτούς επιλεκτικά μια από τις τέσσερις βασικές αριθμητικές πράξεις.

Χρησιμοποιώντας δομημένα Ελληνικά δίνουμε σε πρώτη φάση την περιγραφή της διεργασίας που θα πρέπει να εκτελεί το σύστημα.

1. πάρε τους δύο αριθμούς
2. ενημέρωσε το χρήστη για δυνατές επιλογές
3. πάρε την επιλογή του χρήστη
4. ανάλογα με την επιλογή
5. εκτέλεσε την αντίστοιχη πράξη
6. εμφάνισε το αποτέλεσμα
7. τερμάτισε

Είναι προφανές για σας ότι η γραμμή 1 μπορεί να περιγραφεί σε C με τη χρήση των συναρτήσεων `printf` και `scanf` της βασικής βιβλιοθήκης.

Η γραμμή 2 μπορεί να αποτελεί την κλήση μιας δικής μας συνάρτησης, η οποία θα εμφανίζει στο χρήστη τις δυνατές επιλογές που του δίνει το πρόγραμμα. Αν ονομάσουμε τη συνάρτησή μας `display_menu` είναι προφανές ότι αυτή δε δέχεται κανένα όρισμα και δεν επιστρέφει τιμή. Η δήλωσή της θα έχει τη μορφή:

```
void display_menu();
```

Η συγγραφή του σώματος της συνάρτησης είναι εύκολη δουλειά για σας, όπως και η γραμμή 3 για την οποία υποθέτουμε ότι έχουμε δηλώσει την ακέραια μεταβλητή `choice` για την αναπαράσταση της επιλογής του χρήστη και, ταυτόχρονα, πως για κάθε πράξη αντιστοιχούμε ένα από τους αριθμούς[1] 1, 2, 3 και 4.

Οι γραμμές 4 και 5 είναι προφανές ότι αποτελούν την κλασσική περίπτωση εφαρμογής της πρότασης `switch case`. Η έκφραση με βάση την οποία θα αποφασιστεί η εκτέλεση μιας από τις τέσσερις πράξεις είναι απλή, αποτελείται μόνο από την `choice`. Οι σταθερές δε των ετικετών case θα είναι οι αριθμοί 1, 2, 3 και 4. Μετά από αυτά, η πρόταση `switch` σε ψευδοκώδικα διαμορφώνεται όπως στο σχήμα 7.8(α). Στο σχήμα 7.8(β) όπου δίνεται και ένα τμήμα του αντίστοιχου κώδικα.

[1] Θα μπορούσαμε βέβαια να χρησιμοποιήσουμε άμεσα τα σύμβολα των τεσσάρων πράξεων.

```
switch(choice) {
    case 1: πρόσθεσε τους αριθμούς;          break;
    case 2: αφαίρεσε τους αριθμούς;          break;
    case 3: πολλαπλασίασε τους αριθμούς;     break;
    case 4: διαίρεσε τους αριθμούς;          break;
    default : ενημέρωσε το χρήστη ότι η
              επιλογή δεν υποστηρίζεται;     break;
    }
```

(α) Ψευδοκώδικας

```
switch(choice) {
   case 1:
        result =  num1 + num2;
        break;
   :    /* λοιπά cases */
   default :
        printf("η επιλογή δεν
               υποστηρίζεται\n");
        break;
   }
```

(β) Πηγαίος κώδικας

Σχήμα 7.8: *Παράδειγμα χρήσης* `switch`.

Δραστηριότητα 7.3

Αναπτύξτε το πρόγραμμα του παραδείγματος 3. Εκτελέστε το για να ελέγξετε τη σωστή λειτουργία του. Ζητήστε από το πρόγραμμά σας να διαιρέσει τον αριθμό 10 με το 0. Παρατηρήστε τη συμπεριφορά του συστήματός σας. Τροποποιήστε τον πηγαίο κώδικα ώστε να ενημερώνεται ο χρήστης ότι η δια του μηδενός διαίρεση δεν επιτρέπεται. Τη δική μας έκδοση θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 7.4

Βελτιώστε την αναγνωσιμότητα του πηγαίου κώδικα της προηγούμενης δραστηριότητας χρησιμοποιώντας αλφαριθμητικά της μορφής (ADD, SUB, MUL και DIV) για την αναπαράσταση των επιλογών του χρήστη. Δώστε δύο εκδόσεις, μια με χρήση της εντολής `#define` του προεπεξεργαστή, και δεύτερη με κατάλληλη χρήση του απαριθμητικού τύπου της C. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

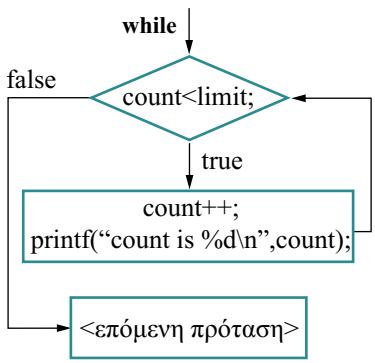
7.2.3 Πρόταση επανάληψης while

Η while ανήκει στην κατηγορία των υπό συνθήκη προτάσεων επανάληψης και αποτελεί την πιο συχνά χρησιμοποιούμενη πρόταση αυτής της κατηγορίας. Η εκτέλεση του σώματος του βρόχου εξαρτάται από την τιμή μιας έκφρασης που υπολογίζεται πριν από την εκτέλεση του σώματος. Έχει δε την παρακάτω σύνταξη:

```
while ( έκφραση )
   πρόταση
```

Υπολογίζεται η τιμή της έκφρασης και, αν είναι αληθής, εκτελείται η πρόταση (απλή ή σύνθετη). Στη συνέχεια, ο έλεγχος μεταφέρεται πάλι στην αρχή της `while`, δηλαδή στον υπολογισμό της τιμής της έκφρασης. Αυτό συνεχίζεται έως ότου η έκφραση τελικά δώσει τιμή ψευδή, οπότε ο έλεγχος μεταφέρεται στην πρόταση που ακολουθεί την `while`. Το σχήμα 7.9 δίνει το διάγραμμα ροής ενός παραδείγματος χρήσης της `while`.

```
while (count<limit) {
   count++;
   printf("count is %d\n",count);
   }
< επόμενη πρόταση>
```



Προσοχή: Εάν η τιμή της `limit` είναι 12 και η τιμή της `count` μεγαλύτερη η ίση του 12, το σώμα του βρόχου δεν εκτελείται ούτε μια φορά.

Σχήμα 7.9

Ροή ελέγχου της πρότασης while.

Παράδειγμα 4

Υπολογισμός μήκους αλφαριθμητικού.

Η συνάρτηση `strlen` της βασικής βιβλιοθήκης υπολογίζει το μήκος του αλφαριθμητικού του οποίου τη διεύθυνση του πρώτου στοιχείου δέχεται σαν όρισμα. Χρησιμοποιεί έναν απαριθμητή τον οποίο αυξάνει μέχρι να συναντήσει τον χαρακτήρα '\0' που σηματοδοτεί το τέλος του αλφαριθμητικού. Προσέξτε στη δεύτερη έκδοση την απλοποίηση της έκφρασης συνθήκης της πρότασης `while` που βασίζεται στο γεγονός ότι κάθε μη 0 είναι αληθές για την C. Στην τρίτη έκδοση προσέξτε την απουσία σώματος για την `while`.

```
int strlen (char str[])
{
int i = 0;
while(str[i]!='\0')
   ++i;
return i;
}
```

α) Πρώτη έκδοση

```
int strlen (char str[])
{
int i = 0;
while(str[i])
   ++i;
return i;
}
```

β) Δεύτερη έκδοση

```
int strlen (char str[])
{
int i = 0;
while(str[i++]);
return i;
}
```

γ) Τρίτη έκδοση

Παράδειγμα 5

Υπολογισμός παραγοντικού.

Το παραγοντικό ενός αριθμού *n* δίνεται από τον τύπο

$n! = n(n-1)(n-2) \ldots (3)(2)(1)$ με $0! = 1$

Εάν αναπαραστήσω το παραγοντικό με την πραγματική μεταβλητή `factorial` και τον ακέραιο αριθμό με τη μεταβλητή `num`, τότε το παρακάτω τμήμα κώδικα υπολογίζει το παραγοντικό του αριθμού `num`:

```
factorial = 1.0;
count = 1;
while(count < num) {
   factorial *= i;
   count++;
   }
```

Δραστηριότητα 7.5

Αναπτύξτε ένα πρόγραμμα που θα δέχεται από το χρήστη έναν ακέραιο, και θα υπολογίζει και εμφανίζει το παραγοντικό του. Εκτελέστε το για να ελέγξετε τη συμπεριφορά του.

Άσκηση Αυτοαξιολόγησης 7.1

Δίνονται οι παρακάτω δυο προτάσεις:

A) `while(++count<12)` Π1;

B) `while(count++ < 12)` Π1;

Περιγράψτε τον τρόπο που ο υπολογιστής τις εκτελεί, εντοπίζοντας τη διαφορά τους, αν υπάρχει.

Άσκηση για παραπέρα εξάσκηση 7.1

Η ακολουθία των αριθμών, όπου ο κάθε αριθμός είναι ίσος με το άθροισμα των δύο προηγουμένων του, είναι γνωστή σαν ακολουθία ή αριθμοί Fibonacci. Οι δύο πρώτοι αριθμοί της ακολουθίας είναι ίσοι με το 1. Αναπτύξτε ένα πρόγραμμα που θα βρίσκει ζητούμενο πλήθος αριθμών Fibonacci. Εκτελέστε το πρόγραμμα και ελέγξτε τη σωστή συμπεριφορά του. Στη συνέχεια, μπορείτε να ανατρέξετε στη σελίδα 254 του [Rojiani '96] για να βρείτε μια έκδοση του προγράμματος.

Δραστηριότητα 7.6

Να γράψετε πρόγραμμα που να απαριθμεί και εμφανίζει τον αριθμό των κενών (spaces) μιας ακολουθίας χαρακτήρων (πρότασης), που θα δέχεται από την κύρια είσοδο. Χρησιμοποιήστε τη συνάρτηση `getchar()` της βασικής βιβλιοθήκης και θεωρήστε τον χαρακτήρα `'\n'` σαν ένδειξη τερματισμού της ακολουθίας. Εκτελέστε το πρόγραμμα και ελέγξτε τη σωστή συμπεριφορά του. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

7.2.4 Πρόταση επανάληψης do while

Η πρόταση do while σαν exit condition loop, σε αντίθεση με την `while`, υπολογίζει την έκφραση και αποφασίζει για την επανάληψη ή όχι του βρόχου, μετά την εκτέλεση του σώματός της. Αυτό σημαίνει πως έχουμε μια τουλάχιστον εκτέλεση του σώματος, ανεξάρτητα από την τιμή της συνθήκης. Το στοιχείο αυτό αποτελεί και τη μόνη διαφορά από την πρόταση `while`. Η σύνταξη της πρότασης είναι:

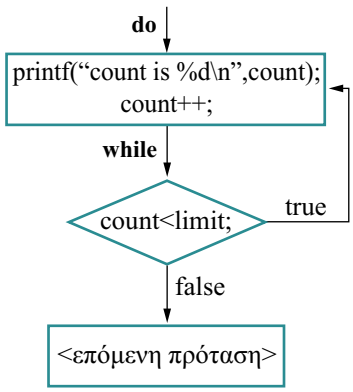
```
do
   πρόταση
   while ( έκφραση );
```

Σχήμα 7.10

Ροή ελέγχου της πρότασης do while.

Το σχήμα 7.10 δίνει το διάγραμμα ροής ενός παραδείγματος χρήσης της `do while`.

```
do {
   count++;
   printf("count is %d\n",count);
   } while (count<limit)
< επόμενη πρόταση>
```



Προσέξτε πως, αν η τιμή της `limit` είναι 12, το σώμα του βρόχου εκτελείται μια φορά έστω και αν η τιμή της `count` είναι μεγαλύτερη ή ίση του 12.

Άσκηση για παραπέρα εξάσκηση 7.2

Θεωρήστε τον πηγαίο κώδικα που αναπτύξατε στη Δραστηριότητα 7.6. Θυμηθείτε τα σχόλιά μας και προσπαθήστε να δώσετε μια έκδοση με τη χρήση της πρότασης do while. Εκτελέστε το πρόγραμμα και ελέγξτε τη σωστή συμπεριφορά του. Στη συνέχεια, μπορείτε να ανατρέξετε στη σελίδα 98 του [Darnell 1991] για να βρείτε μια έκδοση του προγράμματος.

7.2.5 Πρόταση επανάληψης `for`

Η πρόταση αποτελεί αναπαράσταση της ειδικής εκείνης περίπτωσης επανάληψης, στην οποία απαιτείται αρχικοποίηση μιας ή περισσοτέρων μεταβλητών πριν την είσοδο στο βρόχο και, επιπλέον, αλλαγή της τιμής αυτών των μεταβλητών μετά από κάθε εκτέλεση του σώματος του βρόχου.

Η σύνταξη της πρότασης for είναι:

for (E1 ; E2 ; E3) ***ή*** ***for (initialize ; test ; update)***

Π ***πρόταση***

Η εκτέλεσή της περιγράφεται από τα παρακάτω βήματα:

1. Υπολογίζεται η τιμή της έκφρασης *E1*. Η έκφραση, που είναι συνήθως μια έκφραση ανάθεσης, αποδίδει αρχικές τιμές σε μία ή περισσότερες μεταβλητές, απ' όπου και το όνομα *έκφραση αρχικοποίησης*.
2. Υπολογίζεται η τιμή της έκφρασης *E2*, που αποτελεί τη συνθήκη της πρότασης. Εάν η τιμή της είναι ψευδής, ο έλεγχος μεταφέρεται εκτός βρόχου, διαφορετικά εκτελείται η πρόταση *Π*.
3. Μετά την εκτέλεση της πρότασης *Π*, υπολογίζεται η έκφραση *E3*. Η έκφραση αυτή συνήθως μεταβάλει τις τιμές μίας ή περισσοτέρων μεταβλητών, απ' όπου και το όνομα *έκφραση ενημέρωσης* (update). Ο έλεγχος μεταφέρεται πάλι στον υπολογισμό της έκφρασης *E2* (βήμα 2).

Το σχήμα 7.11 δίνει το διάγραμμα ροής ενός παραδείγματος χρήσης της for, ενώ μια πολύ εκτενή αναφορά στην πρόταση for με τους πολλούς εναλλακτικούς τρόπους χρήσης της μπορείτε να βρείτε στο [Waite 90].

Τελεστής κόμμα (,). Ο τελεστής κόμμα (,) επιτρέπει τον υπολογισμό περισσοτέρων της μιας εκφράσεων σε θέσεις όπου επιτρέπετε μια έκφραση. Η τιμή της έκφρασης είναι η τιμή της δεξιότερης έκφρασης. Συνήθως, περιπλέκει τον κώδικα και για το λόγο αυτό η χρήση του είναι περιορισμένη εκτός από την πρόταση for στην οποία συνηθίζεται να χρησιμοποιείται σαν συνθετικό των εκφράσεων αρχικοποίησης και ανανέωσης. Για παράδειγμα, η πρόταση

```
for(i=0, j=10; i<8; i++, j++)
      t[j] = s[i];
```

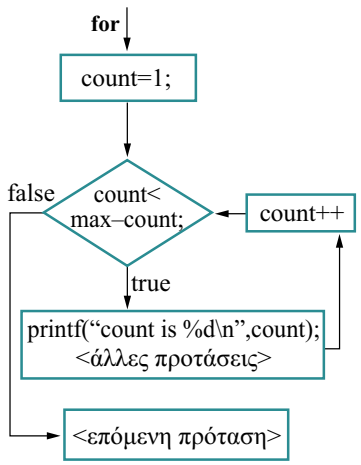
αντιγράφει τα οκτώ πρώτα στοιχεία του πίνακα `s` στον `t` ξεκινώντας από το πρώτο στοιχείο του `s`, το οποίο τοποθετεί στο ενδέκατο στοιχείο του.

Αποφύγετε όμως προτάσεις όπως η

```
for(ch = getchar(), j = 0; ch!=EOF; j++, putchar(ch), ch = getchar())
```

Η πρόταση, αν και είναι συμπαγής ως προς τον κώδικα, μειώνει σε μεγάλο βαθμό την αναγνωσιμότητα του κώδικα. Μια πολύ καλή εφαρμογή του τελεστή κόμμα για το παράδοξο του Ζήνωνα δίνεται στο [Waite 90].

```
for(count=1; count<max_count; count++){
   printf("count is %d\n", count);
   < άλλες προτάσεις >
   }
< επόμενη πρόταση >
```



Σχήμα 7.11

Ροή ελέγχου της πρότασης `for`.

7.2.6 Επιλογή βρόχου

Για την επιλογή του κατάλληλου βρόχου θεωρήστε τις παρακάτω δύο βασικές αρχές:

1. Προτιμήστε το βρόχο συνθήκης εισόδου από τον αντίστοιχο εξόδου, χρησιμοποιήστε το δεύτερο μόνο όταν η φύση του προβλήματος το επιβάλει, υπάρχει δηλαδή ανάγκη εκτέλεσης του σώματος πριν από τον έλεγχο της συνθήκης.
2. Η επιλογή μεταξύ `while` και `for` κρίνεται κυρίως από την ύπαρξη απαριθμητή επαναλήψεων που, συνήθως, συνοδεύεται από αρχικοποίηση και ανανέωση της τιμής του. Στην περίπτωση αυτή, η επιλογή της πρότασης `for` είναι επιβεβλημένη, αν και γενικά ο,τιδήποτε εκφράζεται με τη μια πρόταση μπορεί να εκφραστεί και με την άλλη.

Στη συνέχεια δίνονται ορισμένα παραδείγματα αντιστοιχήσεων

```
for( ; έκφραση ; )
```

```
while(έκφραση)
```

```
for (initialize ; test ; update )
  Π1;
```

```
initialize;
while(test) {
   Π1;
   update;
   }
```

Αναπτύξτε ένα πρόγραμμα που θα δέχεται από το χρήστη έναν ακέραιο αριθμό, και θα υπολογίζει και εμφανίζει το παραγοντικό του. Χρησιμοποιήστε την πρόταση `for`. Εκτελέστε το πρόγραμμα για να ελέγξτε τη συμπεριφορά του. Συγκρίνετε τον κώδικα με αυτόν που δώσατε με πρόταση `while`. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 7.7

Αναπτύξτε ένα πρόγραμμα που θα δέχεται από το χρήστη μια ακολουθία 10 αριθμών και, στη συνέχεια, θα τους τυπώνει με αντίστροφη σειρά. Εκτελέστε το πρόγραμμα για να ελέγξετε την λειτουργία του. Μια έκδοση του ζητούμενου προγράμματος μπορείτε να βρείτε στην σελίδα 142 του [King '96].

Άσκηση για παραπέρα εξάσκηση 7.3

7.2.7 Ένθετοι βρόχοι

Η C δε θέτει κανένα περιορισμό στην ένθεση των προτάσεων ροής ελέγχου, επιτρέποντας στον προγραμματιστή να δημιουργεί ευέλικτες και αποδοτικές κατασκευές. Θυμίζουμε τον κανόνα δόμησης που θέλει το σώμα των βρόχων και των επιλογών να τοποθετούνται στον επόμενο στηλοθέτη.

Να δοθεί η δήλωση και ο ορισμός συνάρτησης που θα δέχεται ένα πίνακα ακεραίων και θα αυξάνει κατά 1 τα θετικά του στοιχεία, θα ελαττώνει δε κατά 2 τα αρνητικά του.

Δραστηριότητα 7.8

Κλασικό παράδειγμα ένθεσης αποτελεί η αναφορά στα στοιχεία ενός πίνακα δύο διαστάσεων. Θεωρήστε τον πίνακα που αναπαριστά τις μέσες θερμοκρασίες των μηνών των τελευταίων 3 ετών. Δώστε α) το βρόχο για τον υπολογισμό των μέσων ετήσιων θερμοκρασιών των τριών ετών β) το βρόχο για τον υπολογισμό των μέσων μηνιαίων θερμοκρασιών των τριών ετών.

Άσκηση Αυτοαξιολόγησης 7.2

Σε συνέχεια της άσκησης Αυτοαξιολόγησης 2 δώστε ένα πρόγραμμα που θα υπολογίζει τις μέσες ετήσιες θερμοκρασίες των τριών ετών, καθώς και τη μέση θερμοκρασία της τριετίας, καθώς και τις μέσες μηνιαίες θερμοκρασίες. Μια δική μας εκδοχή θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 7.9

7.2.8 Προτάσεις break, continue και goto

Οι προτάσεις εξετάστηκαν διεξοδικά στην υποενότητα 7.1.3. Είναι σκόπιμο, τώρα που έχετε ολοκληρώσει τη μελέτη των προτάσεων επανάληψης της C, να μελετήσετε πάλι την παράγραφο 7.1.3. Στη συνέχεια, μπορείτε να ανατρέξετε στη σελίδα 279 του [Rojiani '96] για μια καλή αναφορά στη `break`, και στη σελίδα 111 του [Darnell 1991] για καλά παραδείγματα χρήσης των `break` και `continue`.

Θα τονίσουμε πάλι ότι καλό είναι να αποφεύγετε τη χρήση της `break`. Ο κώδικας που ακολουθεί χρησιμοποιεί τη `break`

```
while (( ch = getchar()) != '\n') {
   if (ch == '\t')
        break; /* στέλνει τον έλεγχο έξω από την while */
   putchar(ch);
   }
```

με αποτέλεσμα να είναι πιο σύνθετος από τον παρακάτω εναλλακτικό

```
while((ch=getchar()) != '\n' &&     ch != '\t')
   putchar(ch);
```

Δραστηριότητα 7.10

Αναπτύξτε ένα πρόγραμμα που θα δέχεται από το χρήστη έναν αριθμό και θα ελέγχει αν ένα τουλάχιστον ψηφίο εμφανίζεται για περισσότερες από μια φορές. Θα τυπώνει δε, το κατάλληλο μήνυμα, π.χ. «επανάληψη ψηφίου». Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

Άσκηση για παραπέρα εξάσκηση 7.4

Θεωρήστε μια σκακιέρα με 100 (10 × 10) θέσεις. Αναπτύξτε ένα πρόγραμμα που θα δημιουργεί μια τυχαία διαδρομή στη σκακιέρα τοποθετώντας τα γράμματα της αλφαβήτου το ένα μετά το άλλο και αρχίζοντας με το A από την πάνω αριστερή θέση. Το πρόγραμμα πρέπει να αποφασίζει τυχαία για το αν θα τοποθετήσει τον επόμενο χαρακτήρα πάνω, κάτω, δεξιά ή αριστερά. Δεν επιτρέπεται τοποθέτηση χαρακτήρα εκτός σκακιέρας αλλά και σε θέση που ήδη είναι κατειλημμένη. Αν η τοποθέτηση του επόμενου χαρακτήρα δεν είναι δυνατή (κατειλημμένες όλες οι γύρω θέσεις), τότε το πρόγραμμα τερματίζει τυπώνοντας το χαρακτήρα εμπλοκής. Χρησιμοποιήστε τη συνάρτηση τυχαίων αριθμών της βασικής βιβλιοθήκης. Για περισσότερα σχόλια μπορείτε να ανατρέξετε στη σελίδα 153 του [King '96].

Σύνοψη

Η διαμόρφωση της ροής εκτέλεσης του προγράμματος είναι από τις σημαντικότερες δουλειές του προγραμματιστή στον προστακτικό προγραμματισμό. Η εξοικείωση με τις διαθέσιμες δομές της γλώσσας διευκολύνει στην επιλογή της κατάλληλης κατασκευής και οδηγεί σε συγγραφή δομημένου, ευανάγνωστου και αποδοτικού κώδικα. Ένα σύνολο από κανόνες διέπουν τη χρήση των προτάσεων ροής ελέγχου:

1. Τοποθετείτε πάντα το σώμα των προτάσεων διακλάδωσης υπό συνθήκη και επανάληψης, μια θέση στηλογνώμονα δεξιότερα. Αυτό αυξάνει την αναγνωσιμότητα του κώδικα. Στην περίπτωση δε που το σώμα αποτελείται από περισσότερες της μιας προτάσεις, περικλείετε απαραίτητα αυτές σε αγκύλες.

2. Αποφεύγετε τη χρήση της πρότασης διακλάδωσης `goto`*. Καταστρέφει τη δόμηση του προγράμματος και τις περισσότερες φορές προδίδει αδυναμία κατασκευής δομημένου κώδικα.*

3. Προτιμήστε το βρόχο επανάληψης συνθήκης εισόδου `(while)` *από τον αντίστοιχο συνθήκης εξόδου* `(do–while)`*, γιατί δίνει πιο ευανάγνωστο κώδικα.*

4. Χρησιμοποιείτε την εντολή break σε προτάσεις `switch` *αλλά, γενικά, αποφεύγετε την χρήση της, όπως και της* `continue` *σε βρόχους επανάληψης. Διακόπτουν την κανονική ροή ελέγχου και καθιστούν την παρακολούθησή της δύσκολη. Προτιμήστε αναδόμηση του κώδικα ή ακόμη και την εισαγωγή μιας σημαίας (flag) που θα αλλάζει τιμή και δεν θα προκαλεί έξοδο «από το παράθυρο» αλλά από την κύρια έξοδο.*

5. Ελέγξτε σχολαστικά και βεβαιωθείτε πως κάθε συνθήκη βρόχου επανάληψης οδηγεί στην έξοδο μετά από πεπερασμένο αριθμό επαναλήψεων.

6. Μια πολύ καλή λίστα συχνά παρατηρούμενων λαθών αλλά και τεχνικών αποφυγής μπορείτε να βρείτε στο [Rojiani '96]

Βιβλιογραφία κεφαλαίου

[Darnell '91]

Darnell P., Margolis P. «*C: A Software Engineering Approach*», Springer–Verlag, New York.

[Rojiani '96]

Rojiani K.B., «*Programming in C with numerical methods for Engineers*», Prentice–Hall.

[Sethi ’97]

Sethi Ravi, «*Programming Languages: Concepts and Constructs*» 2nd Edition, Addison Wesley 1996. Reprinted with corrections April ’97.

[Waite ’90]

Waite Mitchell, Prata Stephen «*New C Primer Plus*» Waite Group Inc.

[Watt ’90]

Watt David, «*Programming Language Concepts and Paradigms*» Prentice Hall.

ΚΕΦΑΛΑΙΟ 8

Προχωρημένα Θέματα Συναρτήσεων

Σκοπός

Σκοπός του κεφαλαίου είναι να παρουσιάσει τους μηχανισμούς μεταβίβασης των παραμέτρων συνάρτησης, την έννοια της αναδρομικότητας στις συναρτήσεις, καθώς και τις έννοιες της ορατότητας και διάρκειας μεταβλητών και του τρόπου που αυτές προσδιορίζονται και επηρεάζουν την οργάνωση του πηγαίου κώδικα.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *δώσετε τον ορισμό της εμβέλειας μεταβλητής,*
- *περιγράψετε τον τρόπο που ορίζεται η κατάλληλη εμβέλεια σε μια μεταβλητή,*
- *ορίσετε μεταβλητές με 3 διαφορετικές εμβέλειες,*
- *περιγράψετε την επίδραση της λέξης–κλειδί* `static` *στη δήλωση μιας γενικής αλλά και μια τοπικής μεταβλητής,*
- *περιγράψετε τη σημασία της διάρκειας μεταβλητής,*
- *οργανώσετε τα προγράμματά σας σε περισσότερα του ενός αρχεία πηγαίου κώδικα,*
- *περιγράψετε τους δύο τρόπους μεταβίβασης παραμέτρων,*
- *αναγνωρίζετε πότε πρέπει να χρησιμοποιήσετε την κατ' αναφορά μέθοδο περάσματος ορισμάτων,*
- *ορίζετε συναρτήσεις χρησιμοποιώντας την κατ' αναφορά μέθοδο μεταβίβασης ορισμάτων,*
- *αναγνωρίζετε πότε μπορεί να χρησιμοποιηθεί η έννοια της αναδρομικότητας στον ορισμό συνάρτησης,*
- *περιγράψετε το μηχανισμό της αναδρομικότητας στις συναρτήσεις,*
- *ορίζετε αναδρομικές συναρτήσεις.*

Έννοιες κλειδιά

- *κανόνες εμβέλειας (scope rules)*
- *εμβέλεια προγράμματος*

- *εμβέλεια αρχείου*
- *εμβέλεια συνάρτησης*
- *εμβέλεια μπλοκ*
- *οργάνωση προγράμματος*
- *αναδρομικότητα (recursion)*
- *γενική μεταβλητή*
- *τοπική μεταβλητή*
- *μεταβίβαση παραμέτρων*
- *κλίση κατά τιμή (call by value)*
- *κλίση κατά αναφορά (call by reference)*
- *static*

Εισαγωγικές Παρατηρήσεις

Στο κεφάλαιο 6 αναφερθήκαμε στις βασικές έννοιες των συναρτήσεων ως δομικών στοιχείων για την ανάπτυξη αρθρωτών προγραμμάτων. Αναφέραμε τον τρόπο δήλωσης και ορισμού συνάρτησης και χρησιμοποιήσαμε τους όρους γενική και τοπική μεταβλητή. Οι όροι αυτοί έχουν σχέση με την εμβέλεια μεταβλητής, την ιδιότητα δηλαδή της μεταβλητής που καθορίζει το μέρος του προγράμματος στο οποίο αυτή είναι ορατή. Μια άλλη, εξίσου σημαντική ιδιότητα της μεταβλητής είναι ο χρόνος που αυτή δεσμεύει τη μνήμη που απαιτείται για αποθήκευση της τιμής της. Οι ιδιότητες αυτές προσδιορίζονται από ορισμένους κανόνες των οποίων η γνώση αποτελεί απαραίτητη προϋπόθεση για την ανάπτυξη ευέλικτων και αξιόπιστων προγραμμάτων.

Το κεφάλαιο αυτό περιλαμβάνει τρεις ενότητες. Η πρώτη ενότητα αναφέρεται στην εμβέλεια των ονομάτων, τη διάρκεια των μεταβλητών και τον τρόπο που οι ιδιότητες αυτές αξιοποιούνται για την οργάνωση του πηγαίου κώδικα. Η δεύτερη ενότητα διαπραγματεύεται το θέμα της μεταβίβασης παραμέτρων. Η μεταβίβαση παραμέτρων με τον τρόπο που τη γνωρίσαμε δεν είναι πάντα σε θέση να υποστηρίξει την επικοινωνία μεταξύ συναρτήσεων. Η μέθοδος της μεταβίβασης κατ' αναφορά, μια από τις εναλλακτικές μεθόδους μεταβίβασης ορισμάτων που υποστηρίζουν οι γλώσσες προστακτικού προγραμματισμού, παρουσιάζεται αναλυτικά. Τέλος, η τρίτη ενότητα αναφέρεται στην έννοια της αναδρομικότητας και του τρόπου με τον οποίο αυτή μπορεί να χρησιμοποιηθεί στον ορισμό συναρτήσεων.

Στο κεφάλαιο αυτό, θα απαιτηθεί να καταβάλλετε ιδιαίτερη προσπάθεια εξ' αιτίας της δυσκολίας των εννοιών που εισάγει. Ιδιαίτερη δυσκολία παρουσιάζει ο μηχανισμός της αναδρομικότητας. Σας συνιστώ να αφιερώσετε περισσότερο χρόνο και να δείξετε περισσότερη επιμονή για την σε βάθος κατανόηση των εννοιών του κεφαλαίου αυτού.

8.1 Εμβέλεια ονομάτων – Διάρκεια μεταβλητών

8.1.1 Εμβέλεια ονομάτων

Είδαμε από το πρώτο κεφάλαιο του συντακτικού ότι ο προγραμματιστής χρησιμοποιεί το λεξιλόγιο της γλώσσας για να δημιουργήσει ονόματα, τα οποία χρησιμοποιεί για να αναφερθεί στις κατασκευές του, δηλαδή τις μεταβλητές, τις συναρτήσεις, τους τύπους δεδομένων και τις σταθερές. Στη C, όπως ήδη γνωρίζετε, η εισαγωγή ενός νέου ονόματος γίνεται με μια πρόταση δήλωσης, η οποία στην ουσία προσδιορίζει και τον τρόπο χρήσης του ονόματος. Μια βασική ερώτηση που μπορεί να τεθεί στο σημείο αυτό είναι «μπορώ να χρησιμοποιήσω το ίδιο όνομα για αναφορά σε δύο ή περισσότερες, διαφορετικές κατασκευές;» και αν ναι «ποιοι είναι οι κανόνες που διέπουν αυτή τη πολλαπλή αντιστοίχηση ενός ονόματος σε περισσότερες κατασκευές;». Στο ερώτημα αυτό απάντηση δίνουν οι *κανόνες εμβέλειας* (scope rules), οι οποίοι προσδιορίζουν την αντιστοίχηση του ονόματος με την κατάλληλη κατασκευή.

Για κάθε εμφάνιση του ονόματος στο πρόγραμμα οι κανόνες αυτοί προσδιορίζουν την κατασκευή στην οποία το όνομα αναφέρεται. Γενικά, οι κανόνες εμβέλειας διακρίνονται σε *λεκτικούς* (lexical) και *δυναμικούς* (dynamic). Σύμφωνα με την πρώτη κατηγορία κανόνων η αντιστοίχηση του ονόματος γίνεται στο χρόνο μεταγλώττισης, ενώ στη δεύτερη περίπτωση, αυτή γίνεται στο χρόνο εκτέλεσης. Οι περισσότερες γλώσσες προγραμματισμού χρησιμοποιούν λεκτικούς κανόνες εμβέλειας[1]. Για μια καλή θεωρητική αναφορά στο θέμα και, μόνο μετά την ολοκλήρωση του παρόντος κεφαλαίου, μπορείτε να ανατρέξετε στο κεφάλαιο 6 «Εμβέλεια και χρόνος δέσμευσης μνήμης» του [Horowitz ’84] ή στην ενότητα 5.3 «Scope Rules for Names» του [Sethi ’97].

Εμείς θα σταθούμε σε ένα πιο βατό ορισμό που συνήθως χρησιμοποιείται από τους C προγραμματιστές. Ο ορισμός αυτός ορίζει σαν κανόνες εμβέλειας τους κανόνες που προσδιορίζουν το τμήμα του πηγαίου κώδικα στο οποίο ένα όνομα είναι ενεργό ή ορατό. Σύμφωνα με τον ορισμό αυτό διακρίνουμε κατά τον Darnell [Darnell ’91]:

1. Εμβέλεια προγράμματος. Μεταβλητές με εμβέλεια προγράμματος είναι γνωστές και σαν γενικές ή καθολικές (global) μεταβλητές. Οι μεταβλητές

[1] Η Lisp είναι μια γλώσσα που παραδοσιακά χρησιμοποιούσε δυναμικούς κανόνες εμβέλειας.

αυτές είναι ορατές από όλες τις συναρτήσεις που απαρτίζουν τον πηγαίο κώδικα, έστω και αν βρίσκονται σε διαφορετικά αρχεία πηγαίου κώδικα. Η εμβέλεια μιας μεταβλητής προσδιορίζεται, κατά κύριο λόγο, από τη θέση της δήλωσης της μεταβλητής και από την ύπαρξη ή μη της λέξης–κλειδί `static`. Έτσι, μεταβλητή που δηλώνεται έξω από μπλοκ και χωρίς τη λέξη–κλειδί `static` έχει εμβέλεια προγράμματος.

2. Εμβέλεια αρχείου. Μεταβλητή με εμβέλεια αρχείου, είναι ορατή μόνο στο αρχείο που δηλώνεται και μάλιστα από το σημείο της δήλωσής της και κάτω. Μεταβλητή που δηλώνεται έξω από μπλοκ με τη λέξη–κλειδί `static` πριν από τον τύπο της, έχει εμβέλεια αρχείου, π.χ. `static int velocity;`

3. Εμβέλεια συνάρτησης. Προσδιορίζει την ορατότητα του ονόματος από την αρχή της συνάρτησης μέχρι το τέλος της. Εμβέλεια συνάρτησης έχουν μόνο οι `goto` ετικέτες.

4. Εμβέλεια μπλοκ (block). Προσδιορίζει την ορατότητα από το σημείο δήλωσης μέχρι το τέλος του μπλοκ στο οποίο δηλώνεται. Το μπλοκ είναι ένα σύνολο από προτάσεις που περικλείονται σε αγκύλες. Μπλοκ είναι η σύνθετη πρόταση, αλλά και το σώμα συνάρτησης. Εμβέλεια μπλοκ έχουν και τα τυπικά ορίσματα των συναρτήσεων.

Η C επιτρέπει τη χρήση ενός ονόματος για την αναφορά σε διαφορετικά αντικείμενα, με την προϋπόθεση ότι αυτά έχουν διαφορετική εμβέλεια, ώστε να αποφεύγεται η σύγκρουση ονομάτων (name conflict). Εάν οι περιοχές εμβέλειας έχουν επικάλυψη, τότε το όνομα με τη μικρότερη εμβέλεια αποκρύπτει (hides) το όνομα με τη μεγαλύτερη.

Δραστηριότητα 8.1

Προσδιορίστε την εμβέλεια κάθε ονόματος του πηγαίου κώδικα του σχήματος 8.1 και προβλέψτε το αποτέλεσμα της εκτέλεσης του προγράμματος, αφού προηγουμένως διορθώσετε ένα λάθος που υπάρχει χωρίς να τροποποιήσετε τις δηλώσεις των μεταβλητών. Χρησιμοποιήστε το μεταγλωττιστή για να σας βοηθήσει στην προσπάθεια εντοπισμού του λάθους. Στη συνέχεια, εκτελέστε το πρόγραμμα για να επιβεβαιώσετε τα αποτελέσματά σας. Αν αυτά δεν συμφωνούν, ανατρέξτε στους κανόνες εμβέλειας που δώσαμε παραπάνω. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

```
1  int max(int a, int b);
2  static void func(int a);
3  int a;
4  static int b;
5  main(){
6  a=12;  b = a--;
7  printf("a: %d\tb: %d \tc: %d \tmax(b+5,a): %d
8         \n", a,b,c,max(b+5,a));
9  func(a+b);
10 }
11 int c = 13;
12 int max(int a, int b){
13 return (a>b?a:b);
14 }
15 static void func(int x){
16 int b=20;
17 printf("a: %d \tb: %d \tc: %d \tx: %d\t
18       max(x,b): %d\n", a,b,c,x,max(x,b));
19 }
```

Σχήμα 8.1: *Εμβέλεια ονομάτων στη C.*

Καθολικές μεταβλητές

Υιοθετήστε το γενικό κανόνα που συνιστά να αποφεύγετε τις καθολικές μεταβλητές. Οι καθολικές μεταβλητές κάνουν το πρόγραμμα δύσκολο στη συντήρηση και κατανόηση επειδή αυξάνουν την πολυπλοκότητά του. Εξάλλου, σαν μοναδικό τους πλεονέκτημα αναφέρεται η δημιουργία ταχύτερου κώδικα που τις περισσότερες φορές δεν είναι ο αυτοσκοπός.

Επιδιώξτε να σας γίνει συνήθεια η χρήση της λέξης–κλειδί `static`. Σας διασφαλίζει από την αναζήτηση πιθανών αναφορών της μεταβλητής σε άλλα αρχεία του προγράμματος αυξάνοντας την αναγνωσιμότητα αλλά και διευκολύνοντας την τροποποίηση του κώδικα. Μια αναλυτική παρουσίαση των υπέρ και κατά των καθολικών μεταβλητών μπορείτε να βρείτε στη σελίδα 188 του [King '96] ή στη σελίδα 355 του [Rojiani '96]

8.1.2 Διάρκεια μεταβλητών

Μια εξίσου σημαντική παράμετρος των μεταβλητών είναι η *διάρκεια* (duration). Η διάρκεια ορίζει το χρόνο που το όνομα της μεταβλητής είναι συνδεδεμένο με τη θέση μνήμης που περιέχει την τιμή της μεταβλητής. Ονομάζουμε *χρόνο δέσμευσης* και *αποδέσμευσης* τους χρόνους που το όνομα συνδέεται με και αποσυνδέεται από τη μνήμη, αντίστοιχα. Έτσι, για τις καθολικές μεταβλητές δεσμεύεται χώρος με την έναρξη της εκτέλεσης του προγράμματος, η δε μεταβλητή σχετίζεται με την ίδια θέση μνήμης μέχρι το τέλος του προγράμματος. Λέμε ότι οι γενικές μεταβλητές είναι *πλήρους διάρκειας*, αντίθετα με τις τοπικές μεταβλητές που είναι *περιορισμένης διάρκειας*. Ο χρόνος δέσμευσης για τις τοπικές μεταβλητές είναι η είσοδος στο μπλοκ όπου είναι δηλωμένη η μεταβλητή, ενώ ο χρόνος αποδέσμευσης είναι

ο χρόνος εξόδου από αυτό. Διαφορετικά, θα μπορούσαμε να το εκφράσουμε, λέγοντας ότι η ανάθεση της μνήμης σε μια τοπική μεταβλητή γίνεται με την είσοδο στο χώρο εμβέλειάς της, η δε αποδέσμευσή της με την έξοδο από αυτόν. Αυτό σημαίνει ότι η τοπική μεταβλητή μιας συνάρτησης δεν διατηρεί την τιμή της από την μία κλήση της συνάρτησης στην επόμενη. Για τις περιπτώσεις όπου κάτι τέτοιο είναι επιθυμητό, η C μας δίνει τη δυνατότητα να τροποποιήσουμε τη διάρκεια μιας τοπικής μεταβλητής από περιορισμένη σε πλήρη, χρησιμοποιώντας τη λέξη–κλειδί static στη δήλωση της μεταβλητής. Στον παρακάτω κώδικα η μεταβλητή `num` είναι τοπική αλλά έχει διάρκεια προγράμματος, αντίθετα με την `temp` η οποία έχει διάρκεια συνάρτησης.

```
func(int x)
{
int temp;
static int num;
:
}
```

Ιδιαίτερη προσοχή πρέπει να δοθεί στην αρχικοποίηση των τοπικών μεταβλητών. Μια τοπική μεταβλητή περιορισμένης διάρκειας αρχικοποιείται, αν βέβαια κάτι τέτοιο έχει οριστεί, με κάθε είσοδο στο μπλοκ στο οποίο ορίζεται. Μια τοπική μεταβλητή πλήρους διάρκειας, αντίθετα, αρχικοποιείται μόνο κατά την ενεργοποίηση του προγράμματος. Ένα καλό παράδειγμα χρήσης τοπικής μεταβλητής πλήρους διάρκειας μπορείτε να βρείτε στην ενότητα 7.1.2. του [Darnell '91] καθώς επίσης και στο κεφάλαιο 13 του [Waite '90], όπου δίνεται η ανάπτυξη της συνάρτησης παραγωγής τυχαίων αριθμών (random number function).

Άσκηση Αυτοαξιολόγησης 8.1

α) Περιγράψτε την επίδραση της λέξης–κλειδί `static` στις δύο δηλώσεις του παρακάτω κώδικα. β) Πότε αρχικοποιείται η μεταβλητή `count` και πότε η `num`;

```
static int num;
void func(int) {
   static int count = 0;
   int num=100;
   …
   }
```

Άσκηση Αυτοαξιολόγησης 8.2

Ο παραπλεύρως πηγαίος κώδικας ορίζει τη συνάρτηση `increment` την οποία καλεί τρεις φορές η `main`. Μελετήστε τον πηγαίο κώδικα και προσδιορίστε την έξοδό του. Στη συνέχεια, αναπτύξτε ένα πρόγραμμα με `main` και `increment`, όπως οι παραπλεύρως. Εκτελέστε το πρόγραμμα για να ελέγξετε αν το αποτέλεσμά σας είναι το σωστό

```
void increment(void);

main()
{
increment();
increment();
increment();
}

void increment(void)
{
int j =2;
static int k =2;
printf("j: %d\tk: %d\n", j++,
        k++");
}
```

8.1.3 Οργάνωση προγράμματος

Ένα C πρόγραμμα αποτελείται από ένα σύνολο από μεταβλητές, συναρτήσεις και ορισμούς τύπων δεδομένων. Η αύξηση του αριθμού των κατασκευών αυτών, ως συνέπεια της πολυπλοκότητας των σημερινών προγραμμάτων, δημιουργεί προβλήματα διαχείρισης. Η χρησιμοποίηση μηχανισμών που βοηθούν στη διαχείριση μεγάλου αριθμού συνθετικών είναι απαραίτητη. Ένας από τους μηχανισμούς αυτούς, η συνάρτηση, μας επιτρέπει να ομαδοποιήσουμε ένα σύνολο από προτάσεις που επιτελούν συγκεκριμένο έργο και, ταυτόχρονα, να αποκρύψουμε στο εσωτερικό τους μεταβλητές (τοπικές μεταβλητές), οι οποίες δεν ενδιαφέρουν τις υπόλοιπες συναρτήσεις του προγράμματος. Ο μηχανισμός αυτός όμως δεν είναι αρκετός από μόνος του για την οργάνωση των μεσαίου και μεγάλου μεγέθους C προγραμμάτων.

Λύση στο πρόβλημα αυτό μας δίνει το αρχείο, το επόμενο επίπεδο αφαιρετικότητας, που εισάγει η C. Μπορείτε να οργανώσετε το πρόγραμμά σας σε περισσότερα του ενός αρχεία, όπου το καθένα θα περιέχει τις κατάλληλες συναρτήσεις και μεταβλητές των οποίων την ορατότητα ρυθμίζετε με την λέξη–κλειδί `static`.

Παράδειγμα 1

Υποθέστε πως το πρόγραμμά σας αποτελείται από τις συναρτήσεις

`main`(), `func1`(), `func2`(), `func3`() και `func4`(). Έχει δε σαν μεταβλητές τις ακέραιες x και y και το αλφαριθμητικό `buf`. Αν, η `main`καλεί τις `func1` και `func2`, η `func1` την `func2`, και η `func2` τις `func3` και `func4` και, επιπλέον, οι `main` και `func1` χρησιμοποιούν την `y`, οι `func2`, `func3` και `func4` την `buf`, και οι `main`, `func2` και `func3` την `x`, τότε οργανώστε το πρόγραμμά σας σε δύο αρχεία πηγαίου κώδικα, ώστε να έχετε τα καλύτερα αποτελέσματα όσον αφορά ορατότητα μεταβλητών και συναρτήσεων.

Το σχήμα 8.2 δίνει έναν ενδεικτικό πηγαίο κώδικα με τις παραπάνω συναρτήσεις και μεταβλητές καθώς και τον τρόπο που αυτές οργανώθηκαν σε δύο αρχεία με ονόματα file1.c και file2.c. Τα σώματα των συναρτήσεων είναι ενδεικτικά και δεν κάνουν χρήση όλων των μεταβλητών που το παράδειγμα ορίζει. Το σχήμα 8.3 δίνει την έξοδο που προκύπτει από την εκτέλεση του προγράμματος που παράγεται από την μεταγλώττιση και σύνδεση των δύο αρχείων file1 και file2.

Στο αρχείο file1.c παρατηρούμε τα εξής: Η γραμμή 3 αποτελεί το πρωτότυπο της συνάρτησης το οποίο απαιτείται από το μεταγλωττιστή για τον έλεγχο τύπων σε κάθε πρόταση κλήσης της συνάρτησης. Το σώμα της `func2` θα βρεί ο συνδέτης στο αρχείο file2.obj. Η γραμμή 4 δηλώνει τη συνάρτηση `func1` με εμβέλεια αρχείου `(static)`. Αυτό σημαίνει πως η func1 μπορεί να κληθεί μόνο από τις συναρτήσεις του file1. Η γραμμή 6 δηλώνει τη μεταβλητή `x` με εμβέλεια προγράμματος. Αυτό σημαίνει πως η `x` είναι ορατή από όλες τις συναρτήσεις του file1, αλλά μπορεί να είναι ορατή και από τις συναρτήσεις του file2, αν στο file2 περιληφθεί η πρόταση 6, η οποία δηλώνει στο μεταγλωττιστή τον τύπο της x για τον έλεγχο τύπων και, επιπλέον, ότι η μεταβλητή είναι δηλωμένη σε άλλο αρχείο (`extern`). Η γραμμή 7 δηλώνει τη μεταβλητή `y` με εμβέλεια αρχείου. Έτσι, η `y` είναι ορατή μόνο από τις συναρτήσεις του file1. Η γραμμή 21 δηλώνει τη μεταβλητή `k` με εμβέλεια μπλοκ, και επιπλέον, ο προσδιοριστής `static` της προσδίδει διάρκεια ζωής προγράμματος σε αντίθεση με την `i` της γραμμής 20 της οποίας η διάρκεια ταυτίζεται με τη διάρκεια της `func1`.

Στο αρχείο file2.c παρατηρούμε τα εξής: Οι γραμμές 3 και 4 δηλώνουν τις συναρτήσεις `func3` και `func4` με εμβέλεια αρχείου. Μπορούν να κληθούν μόνο από τις συναρτήσεις του file2. Η γραμμή 7, δηλώνει την `buf` με εμβέλεια αρχείου, ώστε να μπορεί αυτή να χρησιμοποιηθεί από τις συναρτήσεις του file2, όχι όμως και από αυτές του file1.

file1.c

```
1   #include <stdio.h>
2
3   void func2(void);
4   static int func1(int a);
5
6   int x=1;
7   static int y=13;
8
9   void main()
10  {
11  int i=30;
12
13  printf("y: %d\tfunc(i+1): %d\tfunc1(i): %d
14            \n\n", y, func1(i+1), func1(i));
15  func2();
16  }
17
18  static int func1(int a)
19  {
20  int i=10;
21  static int k=1;
22
23  printf("func1\ta: %d\ti: %d\tk: %d\tx: %d\n"
24            , a, i, k, x++);
24  func2();
26  return(k++);
27  }
```

file2.c

```
1   #include <stdio.h>
2
3   static void func3(void);
4   static void func4(void);
5
6   extern int x;
7   static char buf[20]= "klea";
8
9
10  void func2(void)
11  {
12  printf("func2\tbuf: %s\t", buf); func3();
13  func4();
14  }
15
16  static void func3(void)
17  {
18  printf("func3\tbuf: %s\t", buf);
19  }
20
21  static void func4(void)
22  {
23  printf("func4\tbuf: %s\tx: %d\n\n", buf, x++);
24  }
```

Σχήμα 8.2

Πηγαίος κώδικας Παραδείγματος 1

Ενδιαφέρον παρουσιάζει η πρόταση 13 του file1. Κατ' αρχήν υπολογίζονται οι τιμές των πραγματικών ορισμάτων της `printf`. Καλείται πρώτα η συνάρτηση `func1` με πραγματικό όρισμα την τιμή της `i` (30). Η κλήση αυτή έχει σαν αποτέλεσμα την έξοδο στην οθόνη των γραμμών 1 και 2 του σχήματος 8.3. Στη συνέχεια, καλείται πάλι η `func` αλλά με πραγματικό όρισμα τώρα την τιμή της έκφρασης `i+1`(31). Η κλήση αυτή έχει σαν αποτέλεσμα την έξοδο στην οθόνη των γραμμών 4 και 5 του σχήματος 8.3. Συγκρίνοντας τις εξόδους

των δύο κλήσεων είναι σημαντικό να παρατηρήσουμε την επίδραση της λέξης–κλειδί `static` της γραμμής 21 του file1. Και στις δύο κλήσεις το `i` έχει την αρχική τιμή 10 που του αποδίδεται με κάθε είσοδο στην `func1`, ενώ το `k` παίρνει τιμή μόνο μια φορά και διατηρεί την τιμή από κλήση σε κλήση. Τέλος, η κλήση της `printf` έχει σαν αποτέλεσμα την γραμμή 7 του σχήματος 8.3.

1	func1	a:**30**	i:**10**	k:**1**	x:1	
2	func2	buf:klea	func3	buf:klea	func4	buf:klea x:2
3						
4	func1	a:**31**	i:**10**	k:**2**	x:3	
5	func2	buf:klea	func3	buf:klea	func4	buf:klea x:4
6						
7	y:13	func(i+1):2	func1(i):1			
8						
9	func2	buf:klea	func3	buf:klea	func4	buf:klea x:5

Σχήμα 8.3

Έξοδος προγράμματος Παραδείγματος 1/Κεφ.8

Όπως φαίνεται και από το παράδειγμα 1, μπορείτε να χρησιμοποιείτε το αρχείο για να ομαδοποιείτε συναρτήσεις που χρησιμοποιούν κοινά δεδομένα. Κλασική περίπτωση τέτοιας ομαδοποίησης αποτελεί η υλοποίηση της στοίβας, η οποία περιλαμβάνει ορισμένες μεταβλητές και ένα σύνολο από συναρτήσεις που χρησιμοποιούν τις μεταβλητές αυτές. Στην ενότητα 10.2 του [King '96] μπορείτε να βρείτε ένα τρόπο υλοποίησης της στοίβας.

Η στοίβα μπορεί να χρησιμοποιηθεί για τον υπολογισμό εκφράσεων που ακολουθούν την μεταθεματική σημειογραφία. Ένα χαρακτηριστικό παράδειγμα χρήσης της στοίβας είναι στην ανάπτυξη ενός προγράμματος υπολογισμού εκφράσεων αντίστροφης Πολωνικής σημειογραφίας. Η αντίστροφη Πολωνική σημειογραφία είναι η μεταθεματική σημειογραφία που αναφέραμε στο Κεφάλαιο 5. Σύμφωνα με αυτή, μια έκφραση της γνωστής μας ενθεματικής σημειογραφίας σαν την `(3 + 6 ) * (8 – 6)` παρουσιάζεται με την μορφή `3 6 + 8 6 – *`

Είναι προφανές ότι στη μορφή αυτή, αν γνωρίζουμε τον αριθμό των τελεστέων πάνω στους οποίους ενεργεί ο κάθε τελεστής, δε χρειαζόμαστε παρενθέσεις.

Άσκηση για παραπέρα εξάσκηση 8.1

Αναπτύξτε ένα πρόγραμμα που θα δέχεται σαν είσοδο αριθμητικές εκφράσεις αντίστροφης Πολωνικής σημειογραφίας, θα υπολογίζει τις τιμές τους και θα τις δίνει στο χρήστη. Προχωρήστε στην ανάπτυξη του προγράμματος με την παράλληλη μελέτη της παραγράφου 4.3 του [Kernighan '88]. Στη φάση αυτή δεν είναι απαραίτητο να ασχοληθείτε με την υλοποίηση της `getop`, απλά χρησιμοποιήστε την. Εκτελέστε το πρόγραμμά σας και ελέγξτε την λειτουργικότητά του. Επιμείνετε έως ότου λειτουργήσει σωστά. Στη συνέχεια, οργανώστε το πρόγραμμα έτσι, ώστε να αποτελείται από περισσότερα του ενός αρχεία επιδιώκοντας την καλύτερη δυνατή ορατότητα και διάρκεια μεταβλητών και ορατότητα συναρτήσεων. Αφού ολοκληρώσετε, ανατρέξτε στην παράγραφο 4.4 του ίδιου βιβλίου ή στο ΕΡΜΗΣ C (http://www.ee.upatras.gr/prof/thrambo/C2Java.htm) για να δείτε ένα τρόπο οργάνωσης του πηγαίου κώδικα σε περισσότερα του ενός αρχεία.

8.2 Μεταβίβαση παραμέτρων

Η μεταβίβαση παραμέτρων (parameter passing) στη C γίνεται με δύο τρόπους. Ο πρώτος εξ αυτών είναι γνωστός σαν **κλήση κατά τιμή** (call by value), ενώ ο δεύτερος σαν **κλήση κατ' αναφορά** (call by reference). Στη μεταβίβαση με τιμή η συνάρτηση δουλεύει πάνω σε αντίγραφα των πραγματικών παραμέτρων, ενώ στη μεταβίβαση με αναφορά δουλεύει πάνω στις πραγματικές παραμέτρους. Πριν όμως προχωρήσουμε στην παράθεση των δύο αυτών τρόπων μεταβίβασης ορισμάτων, αφιερώστε λίγο χρόνο για την παρακάτω απλή εκ πρώτης όψεως δραστηριότητα.

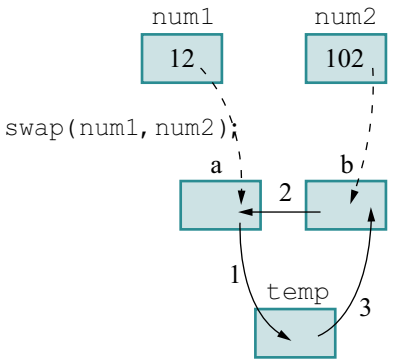
Δραστηριότητα 8.2

Δώστε τη δήλωση και τον ορισμό μιας συνάρτησης η οποία δέχεται δύο ακέραιες μεταβλητές και ανταλλάσσει τις τιμές τους μεταξύ τους. Στη συνέχεια, γράψτε ένα μικρό πρόγραμμα για να δοκιμάσετε τη λειτουργία της συνάρτησής σας. Καταγράψτε τις παρατηρήσεις σας.

Πολλές φορές θέλουμε να ανταλλάξουμε μεταξύ τους τις τιμές δύο μεταβλητών. Πρόκειται για μια απλή διεργασία η οποία θα μπορούσε να αναπαρασταθεί από μία συνάρτηση με το παρακάτω πρωτότυπο

```
void swap(int a, int b);
```

Ο ορισμός της συνάρτησης μπορεί να έχει τη μορφή



Σχήμα 8.4

Μεταβίβαση παραμέτρων με τιμή.

```
void swap(int a, int b) {
int temp;
temp = a;
a = b;
b = temp;
}
```

Τέλος, μια πρόταση κλήσης της συνάρτησης `swap` θα μπορούσε να είναι η

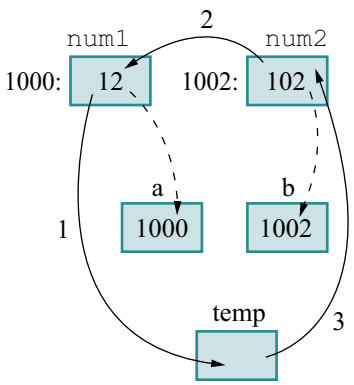
```
swap(num1, num2);
```

για την οποία ας υποθέσουμε ότι οι `num1` και `num2` έχουν τη στιγμή της κλήσης τιμές 12 και 102, αντίστοιχα.

Αν τυπώσουμε πριν την κλήση της `swap` τις `num1` και `num2` θα πάρουμε σαν έξοδο τα 12 και 102. Το ίδιο ακριβώς αποτέλεσμα θα πάρουμε και μετά την κλήση της `swap`. Αυτό σημαίνει πως η `swap` δεν έκανε σωστά την δουλειά της. Και όμως, αν στον κώδικα της swap παρεμβάλω μια πρόταση με την `printf` για να δω τις τιμές των `a` και `b`, θα παρατηρήσω πως οι τιμές τους έχουν αλλάξει. Τι συμβαίνει λοιπόν;

Απάντηση θα μας δώσει ο τρόπος που λειτουργεί ο μηχανισμός κλήσης, ο οποίος περιγράφεται στη συνέχεια. Οι τυπικές παράμετροι `a` και `b` έχουν περιορισμένη διάρκεια που σημαίνει πως δεσμεύεται χώρος στη μνήμη για την αποθήκευση των τιμών τους με την είσοδο στη συνάρτηση. Άρα, δημιουργούνται δύο νέα κελιά στη μνήμη, όπως φαίνεται στο σχήμα 8.4, τα οποία παίρνουν τις τιμές των μεταβλητών `num1` και `num2` που έχουν τα δικά τους κελιά για αποθήκευση των τιμών τους.

Είναι προφανές τώρα ότι η συνάρτηση αλλάζει τις τιμές των κελιών `a` και `b` με τη σειρά που αριθμείται στο σχήμα, χωρίς να επηρεάζει καθόλου τις τιμές των `num1` και `num2`. Το γεγονός αυτό αιτιολογεί πλήρως την εκ πρώτης όψης περίεργη συμπεριφορά της `swap`. Το συμπέρασμα; Πρέπει με κάποιο τρόπο η συνάρτηση να αποκτήσει πρόσβαση στα κελιά των πραγματικών παραμέτρων, ώστε να μπορέσει να αλλάξει τις πρωτότυπες τιμές τους και όχι τις τιμές των αντιγράφων τους. Αυτό μπορεί να γίνει μεταβιβάζοντας στην `swap` τους δείκτες των κελιών των `num1` και `num2` που είναι αντίστοιχα `&num1` και `&num2`, για να θυμηθούμε και τον τελεστή άμεσης διεύθυνσης `&`. Μετά τα παραπάνω η




```
swap(&num1,&num2);
```

Σχήμα 8.5

Πέρασμα παραμέτρων με αναφορά.

κλήση της συνάρτησης θα διαμορφωθεί ως:

```
swap(&num1,&num2);
```

Ανάλογα θα πρέπει να τροποποιηθεί τώρα η δήλωση και ο ορισμός της συνάρτησης. Η δήλωση θα πρέπει να ορίζει την `swap` σαν συνάρτηση που δέχεται δύο δείκτες σε ακέραιους και διαμορφώνεται ως

```
void swap(int *a, int *b);
```

Αντίστοιχα, ο ορισμός της συνάρτησης παίρνει την παρακάτω μορφή:

```
void swap(int *a, int *b) {
int temp;
temp = *a;
*a = *b;
*b = temp;
}
```

Το σχήμα 8.3 αναπαριστά τη διαδικασία κλήσης της swap, υποθέτοντας ότι οι διευθύνσεις των `num1` και `num2` είναι αντίστοιχα 1000 και 1002.

Με την κλήση της `swap` δημιουργούνται δύο νέα κελιά στη μνήμη για τις μεταβλητές δείκτη `a` και `b`. Τα κελιά αυτά παίρνουν σαν τιμές τις διευθύνσεις των μεταβλητών `num1` και `num2`, δηλαδή τα 1000 και 1002, όπως φαίνεται και από τις διακεκομμένες γραμμές του σχήματος 8.5. Έτσι, η συνάρτηση `swap` έχοντας τις διευθύνσεις των `num1` και `num2` αλλάζει τις τιμές τους με τη σειρά που αριθμείται στο σχήμα. Η πρώτη πρόταση `temp = *a;` της swap λέει «απόδωσε στην `temp` την τιμή που είναι αποθηκευμένη στη διεύθυνση 1000» (δες βέλος 1). Η `*a = *b;` αποδίδει την τιμή της `num2` στην `num1` και, τέλος, η `*b = temp;` αποδίδει την τιμή της `temp` στην `num2` ολοκληρώνοντας το έργο της συνάρτησης.

Προσέξτε ιδιαίτερα τη μεταβίβαση μεταβλητής τύπου πίνακα. Είναι ο μόνος τύπος που δεν μπορεί να μεταβιβαστεί με τιμή, αλλά μεταβιβάζεται πάντα με αναφορά.

Για περισσότερες πληροφορίες σχετικά με τις μεθόδους μεταβίβασης παραμέτρων, μπορείτε να ανατρέξετε στην ενότητα 5.2 του [Sethi ’97] όπου παρουσιάζονται οι πιο συχνά χρησιμοποιούμενες μέθοδοι call–by–value, call–by–reference και η παραλλαγή τους call–by–value–result ανεξαρτήτως γλώσσας προγραμματισμού. Ιστορικής σημασίας και σπάνια χρησιμοποιούμενη είναι πλέον η μέθοδος call–by–name. Επίσης, στο τελευταίο τμήμα της

ενότητας 6.1 του [Horowitz '84] μπορείτε να ενημερωθείτε για τον τρόπο με τον οποίο οι σημαντικές γλώσσες προστακτικού προγραμματισμού αντιμετώπισαν το θέμα της μεταβίβασης παραμέτρων και διαμόρφωσαν τις κατάλληλες μεθόδους.

Δραστηριότητα 8.3

Αναπτύξτε ένα μικρό πρόγραμμα που θα επιδεικνύει τη χρήση της συνάρτησης swap. Εκτελέστε το για να επιβεβαιώσετε τη λειτουργία του.

Άσκηση Αυτοαξιολόγησης 8.3

Χρησιμοποιώντας μεταβίβαση με αναφορά, δώστε τη δήλωση, τον ορισμό και ένα παράδειγμα κλήσης της συνάρτησης εύρεσης του μέγιστου μεταξύ δύο ακέραιων αριθμών.

8.3 Αναδρομικότητα

Εισαγωγικές Παρατηρήσεις

Η αναδρομικότητα (recursion) αποτελεί μια πολύ ενδιαφέρουσα προγραμματιστική τεχνική που μας δίνει αποτελεσματικές λύσεις για μια ευρεία κατηγορία προβλημάτων. Πριν προχωρήσετε στη μελέτη της ενότητας που είναι από τις πιο δύσκολες του βιβλίου, σας συνιστώ να εκτελέσετε την παρακάτω απλή δραστηριότητα. Θα σας βοηθήσει σημαντικά στην κατανόηση της τεχνικής της αναδρομής σαν διαφορετικού τρόπου σκέψης για την επίλυση ενός προβλήματος. Σε κάθε περίπτωση, όμως, η όποια δυσκολία αντιμετωπίσετε είναι απόλυτα δικαιολογημένη, μην σας απογοητεύσει.

Δραστηριότητα 8.4

Δώστε τη δήλωση και τον ορισμό μιας συνάρτησης που υπολογίζει το άθροισμα των αριθμών από 1 μέχρι `n`.

Η προφανής δήλωση της συνάρτησης θα είναι η `int sum(int n);` που ορίζει τη συνάρτηση με όνομα `sum` και μοναδικό όρισμα τον ακέραιο *n*. Για τον ορισμό του σώματος της συνάρτησης, ο κλασικός τρόπος σκέψης μας οδηγεί στη δήλωση μιας μεταβλητής `total`, `(int total=0;)` και στη δημιουργία ενός βρόχου από 1 έως, *n* οπότε και το σώμα της `sum` διαμορφώνεται όπως παρα-

κάτω.

```
for(i=0; i=<n; i++)
        total +=n;
return total;
```

Προσέξτε τώρα ένα διαφορετικό τρόπο αντιμετώπισης. Ο υπολογισμός του αθροίσματος *n* μπορεί να θεωρηθεί σαν υπολογισμός του αθροίσματος των αριθμών μέχρι το *n*–1 συν το *n*. Ο υπολογισμός του αθροίσματος των αριθμών μέχρι τον *n*–1 μπορεί να γίνει από την ίδια συνάρτηση, αρκεί να της περάσουμε το κατάλληλο όρισμα δηλαδή το *n*–1. Έτσι, οδηγούμαστε σε μια πρόταση της μορφής

```
return (sum(n–1) + n);
```

Τι σημαίνει όμως αυτό; Η συνάρτηση sum καλεί τον ίδιο της τον εαυτό, με διαφορετικό όμως πραγματικό όρισμα τώρα. Η κλήση αυτή προκαλεί μια νέα ενεργοποίηση της συνάρτησης sum με το δικό της τυπικό όρισμα, το οποίο δεν έχει καμμία σχέση με το αντίστοιχο όρισμα της προηγούμενης ενεργοποίησης. Η τιμή του τυπικού ορίσματος *n* είναι κατά ένα μικρότερη από την τιμή του τυπικού ορίσματος της προηγούμενης ενεργοποίησης. Η ενεργοποίηση αυτή όταν θα ολοκληρωθεί θα επιστρέψει το άθροισμα των αριθμών μέχρι το *n*–1. Την επιστρεφόμενη αυτή τιμή της πρώτης ενεργοποίησης της η sum θα προσθέσει με το *n* και θα επιστρέψει τελικά το ζητούμενο αποτέλεσμα.

Ας επανέλθουμε όμως στην εκτέλεση της δεύτερης ενεργοποίησης. Η ενεργοποίηση αυτή θα προκαλέσει με τη σειρά της μια νέα ενεργοποίηση και αυτό θα συνεχιστεί μέχρι εμείς να το διακόψουμε. Αυτό το τελευταίο είναι πολύ σημαντικό και είναι αποκλειστική ευθύνη του προγραμματιστή. Στην περίπτωση της sum θα πρέπει να τερματίσουμε τη διαδικασία κλήσης του εαυτού της όταν το τυπικό της όρισμα *n* πάρει την τιμή 1, οπότε και ο υπολογισμός του αθροίσματος είναι απλός και δεν απαιτείται παραπέρα κλήση της sum. Έτσι, η συνάρτηση sum, που ονομάζεται **αναδρομική συνάρτηση**, όπως καλούνται οι συναρτήσεις που καλούν τον εαυτό τους, διαμορφώνεται όπως παρακάτω:

```
int sun(int n)
{
if(n =< 1)
   return n;
else
   return (sum(n–1) + n);
}
```

Αν και η αναδρομικότητα δεν παρέχει ταχύτερο κώδικα και μπορεί να δημιουργήσει stack overflow σε περίπτωση κακής χρήσης, δίνει συνήθως πιο συμπαγή κώδικα και είναι ιδιαίτερα χρήσιμη για αναδρομικώς ορισμένα δεδομένα όπως οι λίστες και τα δέντρα.

Παράδειγμα 2

Ας θεωρήσουμε πως δεν διαθέτουμε τη συνάρτηση `printf` αλλά μόνο την `putchar` της βασικής βιβλιοθήκης, και θέλουμε να ορίσουμε μια συνάρτηση που να εμφανίζει ακέραιους αριθμούς.

Ονομάζουμε `printd` τη συνάρτηση και *n* την τυπική παράμετρο που δέχεται. Για τη δημιουργία του σώματος της συνάρτησης σκεφτόμαστε όπως παρακάτω. Ας υποθέσουμε πως θέλουμε την εκτύπωση του αριθμού 1821. Η διεργασία αυτή μπορεί να αποσυντεθεί στην εκτύπωση του αριθμού 182 που προκύπτει σαν τιμή της έκφρασης `1821/10` και, στη συνέχεια, του ψηφίου 1 που προκύπτει σαν τιμή της έκφρασης `1821%10`. Η εκτύπωση του 1 είναι απλή, καλούμε την `putchar('1')` ή, πιο γενικά, `putchar(1821%10)` ή, ακόμη πιο γενικά, `putchar(n%10)` όπου το *n* αναπαριστά την τυπική παράμετρο της `printd`.

Η εκτύπωση τώρα του 182 μπορεί με τη σειρά της να θεωρηθεί σαν εκτύπωση του 18 (182/10) και, στη συνέχεια, του 2 (182%10). Η γενική μορφή που πρέπει να έχει η πρόταση που καλεί τον εαυτό της είναι `printd(n/10);` Δεν θα πρέπει βέβαια να παραλείψουμε τον έλεγχο για τον τερματισμό της αναδρομής και ο κώδικάς μας ολοκληρώθηκε.

```
/* printing a number as a character string */
#include <stdio.h>
void printd(int n)    /* print n in decimal */
{
   int i;

   if(i<0) {
        putchar('-');
        n = -n;
        }
   if((i = n/10) != 0)
        printd(i);
   putchar(n%10 + '0');
}
```

Άσκηση Αυτοαξιολόγησης 8.4

Θεωρήστε την κλήση της συνάρτησης `sum` με το 5 σαν τιμή πραγματικής παραμέτρου. Συμπληρώστε έναν πίνακα με τον αριθμό των ενεργοποιήσεων της συνάρτησης και για κάθε ενεργοποίηση να δίνετε την τιμή της τυπικής παραμέτρου και την επιστρεφόμενη τιμή.

Δραστηριότητα 8.5

Δώστε τη δήλωση και τον ορισμό μιας αναδρομικής συνάρτησης για τον υπολογισμό του x^n. Γράψτε ένα πρόγραμμα που να διαβάζει ένα πραγματικό αριθμό `x` και τον ακέραιο `n` και, στη συνέχεια, να καλεί την αναδρομική συνάρτησή σας για τον υπολογισμό του x^n το οποίο και να τυπώνει. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

Δραστηριότητα 8.6

Δώστε τη δήλωση και τον ορισμό της αναδρομικής συνάρτησης για τον υπολογισμό του *n*–οστού αριθμού Fibonacci. Γράψτε ένα πρόγραμμα που να διαβάζει έναν ακέραιο θετικό αριθμό *n*, να καλεί τη συνάρτηση υπολογισμού του αριθμού Fibonacci και να τυπώνει το αποτέλεσμα. Τα δικά μας σχόλια θα βρείτε στο τέλος του βιβλίου.

Σύνοψη

Στο κεφάλαιο αυτό αναφερθήκαμε σε πιο προχωρημένες έννοιες των συναρτήσεων. Αναφερθήκαμε στην εμβέλεια ονομάτων και στη διάρκεια μεταβλητών, καθώς και τους κανόνες που προσδιορίζουν τα δύο αυτά χαρακτηριστικά, που είναι ιδιαίτερης σημασίας στην ανάπτυξη μεσαίου και μεγάλου μεγέθους προγραμμάτων. Εξηγήσαμε γιατί ο κατά τιμή μηχανισμός μεταβίβασης ορισμάτων δεν είναι αρκετός για να καλύψει τις ανάγκες επικοινωνίας των συναρτήσεων και παρουσιάσαμε την μεταβίβαση με αναφορά σαν εναλλακτική μέθοδο επικοινωνίας. Θα πρέπει όμως να είστε ιδιαίτερα προσεκτικοί όταν χρησιμοποιείτε τη δεύτερη αυτή μέθοδο, γιατί η καλούμενη συνάρτηση έχει άμεση πρόσβαση πάνω στα πραγματικά ορίσματα και όχι σε αντίγραφά τους. Είδαμε, τέλος, πως η χρήση της αναδρομικότητας στον ορισμό συναρτήσεων διευκολύνει σε ορισμένες περιπτώσεις παρέχοντας πιο συμπαγή κώδικα, αλλά πάντα με την ευθύνη του προγραμματιστή για τερματισμό της αναδρομής.

Βιβλιογραφία

[Darnell 1991]

Darnell P., Margolis P. «*C: A Software Engineering Approach*», Springer–Verlag, New York.

[Horowitz '84]

«*Βασικές αρχές γλωσσών προγραμματισμού*» Εκδόσεις Κλειδάριθμος 1993.

[Kernighan 88]

Η δεύτερη έκδοση του βιβλίου «*The C Programming Language*» κυκλοφόρησε το 1988. καλύπτοντας πλέον την ANSI C. Ελληνική έκδοση σε μετάφραση από τον Κλειδάριθμο 1990.

[King '96]

King K.N., «*C Programming: A modern approach*», W.W.Norton & Company, Inc.

[Rojiani '96]

Rojiani K.B., «*Programming in C with numerical methods for Engineers*», Prentice–Hall.

[Sethi '97]

Sethi Ravi, «*Programming Languages: Concepts and Constructs*» 2nd Edition, Addison Wesley 1996. Reprinted with corrections April '97.

[Waite '90]

Waite Mitchell, Prata Stephen «*New C Primer Plus*» Waite Group Inc.

ΚΕΦΑΛΑΙΟ 9

Αφαιρετικότητα στα Δεδομένα

Σκοπός

Σκοπός του κεφαλαίου είναι να εισάγει την έννοια της αφαιρετικότητας στα δεδομένα και τον τρόπο με τον οποίο η κατασκευή της δομής υποστηρίζει την εφαρμογή της στον προγραμματισμό.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *δώσετε τον ορισμό της αφαιρετικότητας στα δεδομένα,*
- *ορίζετε δομές κατάλληλες για την αναπαράσταση των δεδομένων του προβλήματός σας,*
- *αναφέρεστε στα μέλη δομής,*
- *δηλώνετε αλλά και να αποδίδετε αρχικές τιμές σε μεταβλητές δομών που έχετε ορίσει,*
- *χρησιμοποιείτε την ένθεση δομών για τον ορισμό σύνθετων τύπων,*
- *χρησιμοποιείτε τις δομές για τη μεταβίβαση πληροφορίας προς και από συναρτήσεις,*
- *χρησιμοποιείτε τον τύπο της δομής σε συνδυασμό με το συναθροιστικό τύπο του πίνακα.*

Έννοιες κλειδιά

- *αφαιρετικότητα στα δεδομένα*
- *δομή*
- *μέλος δομής*
- *ένθεση δομών*
- *ορισμός δομής*
- *struct*

Εισαγωγικές Παρατηρήσεις

Όπως ήδη γνωρίζετε, μια από τις βασικότερες τεχνικές με τις οποίες ο άνθρωπος αντιμετωπίζει την πολυπλοκότητα είναι η αφαιρετικότητα (abstraction), η οποία εμφανίζεται με δύο μορφές: α) αφαιρετικότητα στις διεργασίες και β) αφαιρετικότητα στα δεδομένα. Η αφαιρετικότητα στα δεδομένα δεν είναι καινούργια έννοια, τη χρησιμοποιούμε συχνά στην καθημερινή μας ζωή. Στη

φράση «ποια είναι η διεύθυνσή σου» χρησιμοποιούμε τη λέξη «διεύθυνση» για να αναφερθούμε στην πληροφορία που προσδιορίζει την οδό, τον αριθμό, τον ταχυδρομικό κώδικα και την πόλη.

Οι σχεδιαστές γλωσσών προγραμματισμού διαπίστωσαν νωρίς την ανάγκη παροχής μηχανισμού για την υλοποίηση της αφαιρετικότητας στα δεδομένα. Ένα πρώτο βήμα προς την κατεύθυνση αυτή αποτέλεσαν οι τύποι (types) της Pascal και τα modes της ALGOL68. Όμως, η SIMULA67 ήταν αυτή που έδωσε ένα πιο πλήρη μηχανισμό για την υλοποίηση της έννοιας. Ο μηχανισμός της κλάσης τον οποίο εισήγαγε, αποτέλεσε τη βάση για την υλοποίηση της αφαιρετικότητας από τις γλώσσες αντικειμενοστρεφούς προγραμματισμού, καθώς επιτρέπει την ολοκλήρωση ενός συνόλου δεδομένων με τις πράξεις που είναι δυνατές πάνω σε αυτά τα δεδομένα.

Στη θεματική αυτή ενότητα θα αφιερώσουμε σχετικά λίγο χώρο για την περιγραφή του μηχανισμού με τον οποίο η C υποστηρίζει την αφαιρετικότητα στα δεδομένα, καθώς η έννοια υποστηρίζεται πολύ καλύτερα από τις αντικειμενοστρεφείς γλώσσες προγραμματισμού, όπως θα δείτε στη Θεματική Ενότητα 6.4. Η C χρησιμοποιεί την κατασκευή struct, που είναι ανάλογη της record της Pascal, και επιτρέπει τη συνάθροιση δεδομένων διαφορετικών τύπων για τον ορισμό ενός νέου τύπου. Στη συνέχεια, βέβαια, χρησιμοποιείται η κατασκευή της συνάρτησης για τον ορισμό των λειτουργιών που επιτρέπονται πάνω στο νέο αυτό τύπο.

9.1 Η έννοια της δομής

Χρησιμοποιήσαμε τους τύπους δεδομένων για να προσδιορίσουμε τις μεταβλητές με τις οποίες αναπαριστούμε τα δεδομένα του προβλήματός μας. Έτσι, προσδιορίσαμε σαν ακέραιου τύπου τη μεταβλητή του απαριθμητή, και σαν τύπου κινητής υποδιαστολής απλής ακρίβειας τη μεταβλητή που αναπαριστά τη θερμοκρασία. Τι γίνεται όμως με την αναπαράσταση δεδομένων όπως «διεύθυνση», «συντεταγμένες τετραγώνου» κ.λπ. Προφανώς, είναι αδύνατο να έχουμε έναν τύπο για την αναπαράσταση κάθε έννοιας του φυσικού προβλήματος. Οι σύγχρονες γλώσσες προγραμματισμού για να επιτρέψουν το χειρισμό ομάδων δεδομένων διαφορετικού τύπου, τα οποία έχουν σχέση μεταξύ τους, παρέχουν ένα μηχανισμό που επιτρέπει τον ορισμό νέων τύπων.

Ο συναθροιστικός τύπος της *δομής* (structure) ή της *εγγραφής* (record) της Pascal, σε αντίθεση με τον άλλο συναθροιστικό τύπο, αυτόν του πίνακα, μας επιτρέπει να χειριζόμαστε ομάδες δεδομένων, τα οποία έχουν κάποια σχέση μεταξύ τους και είναι διαφορετικού τύπου. Τον μηχανισμό `struct` χρησιμοποιούμε για να ορίσουμε ένα νέο τύπο, είτε όταν θέλουμε να αναπαραστήσουμε μια έννοια ή ένα αντικείμενο του φυσικού προβλήματος που χαρακτηρίζεται από ένα σύνολο ιδιοτήτων διαφορετικού πιθανόν τύπου ή όταν για οποιοδήποτε λόγο θέλουμε να ομαδοποιήσουμε κάτω από ένα όνομα μεταβλητές του προγράμματος οι οποίες είναι πιθανόν διαφορετικού τύπου. Επιπλέον, σε αντίθεση με τον πίνακα, όπου το κάθε διαφορετικό δεδομένο προσδιορίζεται από την τάξη του μέσα σε αυτόν, το κάθε δεδομένο μέλος της δομής, που ονομάζεται και *πεδίο* (field), μπορεί να έχει το δικό του όνομα. Τα μέλη μιας δομής μπορεί να ανήκουν στους βασικούς τύπους `int`, `char`, `float` και `double`, μπορεί να είναι πίνακες αλλά ακόμη και άλλες δομές. Για να ορίσουμε μεταβλητές τύπου δομής πρέπει πρώτα να ορίσουμε τη δομή.

9.2 Ορισμός δομής

Ο ορισμός μιας δομής έχει την παρακάτω μορφή:

```
struct [<όνομα δομής >] {
    <τύπος 1–ου μέλους> <όνομα 1–ου μέλους>;
    <τύπος 2–ου μέλους> <όνομα 2–ου μέλους>;
    <τύπος 3–ου μέλους> <όνομα 3–ου μέλους>;
    …
```

<τύπος n–ου μέλους> <όνομα n–ου μέλους>;
};

Εάν δύο ή περισσότερα μέλη έχουν τον ίδιο τύπο, η αναφορά τους γίνεται με πιο απλοποιημένη μορφή

<τύπος μέλους> <όνομα 1ου μέλους>, <όνομα 2ου μέλους>;

Ο ορισμός της δομής ορίζει στην πράξη ένα «καλούπι», σύμφωνα με το οποίο θα δεσμεύεται μνήμη για τις δηλούμενες μεταβλητές της νέας δομής ή αλλιώς, του νέου τύπου. Σαν παράδειγμα, δίνεται ο ορισμός της δομής ορθογώνιο (rectangle):

```
struct rectangle {
   int x1, y1;        /* συντεταγμένες κάτω αριστερής γωνίας */
   int x2, y2;        /* συντεταγμένες πάνω δεξιάς γωνίας */
   int line_color;
   int fill_color;
   };
```

9.3 Δήλωση μεταβλητών

Η δήλωση μεταβλητών ακολουθεί το γενικό κανόνα δηλώσεων της C και γίνεται με πρόταση δήλωσης της μορφής:

`struct` **<όνομα δομής> <όνομα μεταβλητής>;**

ή

`struct` **<όνομα δομής> λίστα–ονομάτων–μεταβλητών ;**

Εναλλακτικά, μπορεί να γίνει δήλωση μεταβλητής ή μεταβλητών ταυτόχρονα με τον ορισμό της δομής όπως παρακάτω:

struct [<όνομα δομής >] {
<τύπος 1ου μέλους> <όνομα 1ου μέλους>;
<τύπος 2ου μέλους> <όνομα 2ου μέλους>;
<τύπος 3ου μέλους> <όνομα 3ου μέλους>;
…
<τύπος nου μέλους> <όνομα nου μέλους>;
} λίστα–ονομάτων–μεταβλητών;

Για παράδειγμα, για τη δήλωση των μεταβλητών rect1 και rect2 μπορούμε να γράψουμε:

```
struct rectangle {
   int x1, y1; /* συντεταγμένες κάτω αριστερής γωνίας */
   int x2, y2; /* συντεταγμένες πάνω δεξιάς γωνίας */
   int line_color;
   int fill_color;
   } rect1, rect2;
```

ή, πιο απλά

```
struct rectangle rect1, rect2;
```

με την προϋπόθεση ότι έχει προηγηθεί ο ορισμός της δομής `rectangle`.

Η σωστή δόμηση του προγράμματος επιβάλλει το διαχωρισμό του ορισμού της δομής από τις δηλώσεις των αντίστοιχων μεταβλητών.

9.4 Απόδοση αρχικών τιμών

Μπορούμε να αποδώσουμε αρχικές τιμές στις μεταβλητές τη στιγμή της δήλωσής τους. Η απόδοση αυτή μπορεί να γίνει μαζί με τον ορισμό και τη δήλωση, όπως παρακάτω:

```
struct rectangle {
   int x1, y1; /* συντεταγμένες κάτω αριστερής γωνίας */
   int x2, y2; /* συντεταγμένες πάνω δεξιάς γωνίας */
   int line_color;
   int fill_color;
   }    rect1= {10, 10,  40, 40, RED, GREEN},
        rect2 = {5, 5, 50, 50, RED, BLUE};
```

ή, με τη δήλωση, όπως παρακάτω:

```
struct rectangle rect1= {10, 10,  40, 40, RED, GREEN},
                 rect2 = {5, 5, 50, 50, RED, BLUE};
```

Αναπαραστήστε δύο κύκλους με κέντρο (2.0,3.0) και ακτίνα 1.0 και κέντρο (4.0,6.0) και ακτίνα 2.0, αντίστοιχα.

Άσκηση Αυτοαξιολόγησης 9.1

Αναπαραστήστε τη διεύθυνσή σας με μία μεταβλητή.

Άσκηση Αυτοαξιολόγησης 9.2

9.5 Αναφορά στα μέλη δομής

Η αναφορά στα μέλη δομής γίνεται με κατάλληλο συνδυασμό του ονόματος της μεταβλητής και του ονόματος του μέλους. Χρησιμοποιείται ο τελεστής τελεία (.) για να σχηματισθεί η έκφραση της παρακάτω μορφής:

<όνομα μεταβλητής>.<όνομα μέλους>

Έτσι, η έκφραση

`rect1.x1`

αναφέρεται στο μέλος `x1` της μεταβλητής `rect1`, ενώ η

`rect2.line_color`

αναφέρεται στο μέλος `line_color` της μεταβλητής `rect2`.

9.6 Ένθεση δομών

Μια δομή μπορεί να περιλαμβάνει μέλη τα οποία είναι τύπου άλλης δομής. Η C δεν βάζει κανένα περιορισμό στο βαθμό ένθεσης. Η δομή `person`, ο ορισμός της οποίας ακολουθεί, έχει το μέλος `addr` που είναι τύπου `address` αλλά και το μέλος `birthday` που είναι τύπου `day`.

```
struct person {
   …    /* λοιπά μέλη δομής */
   struct address addr;
   struct day birthday;
   };
```

Βέβαια, θα πρέπει να έχουν προηγηθεί οι ορισμοί των δομών `address` και `day`.

Η αναφορά στο μέλος `city` της διεύθυνσης για τη μεταβλητή `emp` που είναι τύπου `person` γίνεται με την έκφραση

`emp.addr.city`

9.7 Πέρασμα δομής σε συνάρτηση

Η ANSI C υποστηρίζει την κατ' αναφορά αλλά και την κατά τιμή μεταβίβαση των δομών στις συναρτήσεις. Έτσι, είναι επιλογή του προγραμματιστή, αν θα περάσει επιλεκτικά ορισμένα μέλη της δομής, θα περάσει όλη τη δομή ή θα περάσει μόνο ένα δείκτη σε αυτή.

Άσκηση Αυτοαξιολόγησης 9.3

Χρησιμοποιήστε την ένθεση δομών για την καλύτερη αναπαράσταση του κύκλου. Στη συνέχεια, γράψτε μια συνάρτηση, η οποία θα επιτρέπει στο χρήστη να αποδίδει τιμές στα πεδία μεταβλητών τύπου κύκλου. Δώστε ένα παράδειγμα κλήσης της.

9.8 Πίνακες δομών

Η δήλωση πίνακα δομών έχει ως κατωτέρω:

```
struct address addr[10];
```

Η απόδοση αρχικής τιμής στη δομή είναι ανάλογη με το γενικό κανόνα αρχικοποίησης πινάκων. Έτσι, για την αρχικοποίηση των τριών πρώτων στοιχείων του πίνακα `addr` έχουμε την παρακάτω δήλωση–αρχικοποίησης:

```
struct address addr[10] =    {
                             {"Kleanthis", "Telou Agra", 10, 24662,
                             "Patras" },
                             {"Nikos","Marathona",12,12345,
                             "Athens" },
                             {"kostas","Korinthou",340, 24561,
                             "Aigio" }
                             };
```

Τα υπόλοιπα στοιχεία του πίνακα `addr` θα έχουν μηδενικές τιμές.

Για την αναφορά στα μέλη των στοιχείων του πίνακα, ακολουθείται η προφανής σύνταξη. Η έκφραση

```
addr[0].name
```

αναφέρεται στο όνομα της πρώτης διεύθυνσης του πίνακα, ενώ η

```
addr[1].name
```

αναφέρεται στο όνομα της δεύτερης διεύθυνσης του πίνακα.

Προσέξτε την έκφραση

```
addr[1].name[0]
```

αναφέρεται στον πρώτο χαρακτήρα τού ονόματος τού πρώτου στοιχείου τού πίνακα `addr`.

9.9 Δείκτες σε δομές

Όπως κάθε μεταβλητή, έτσι και η μεταβλητή `addr1` που ορίζεται από τη δήλωση

```
struct address addr1;
```

έχει διεύθυνση την οποία μπορούμε να πάρουμε εφαρμόζοντας στη μεταβλητή τον τελεστή `&`. Έτσι, η έκφραση `&addr1` δίνει τη διεύθυνση της μεταβλητής `addr1`.

Αν δηλώσω ένα δείκτη σε δομή `address`, μπορώ να τον βάλω να δείχνει στην `addr1` με την παρακάτω πρόταση:

```
struct address *addr_ptr = &addr1;
```

Μετά από τη δήλωση αυτή ο δείκτης `addr_ptr` δείχνει στην `addr1`, παρέχοντάς μας ένα εναλλακτικό τρόπο πρόσβασης στα μέλη της. Η έκφραση

```
addr_ptr->name
```

αναφέρεται στο μέλος `name` της δομής που δείχνει ο δείκτης `addr_ptr`, ενώ η

```
addr_ptr->zip
```

αναφέρεται στο μέλος `zip` της δομής που δείχνει ο δείκτης `addr_ptr`.

Οι δείκτες σε δομές βρίσκουν ιδιαίτερη εφαρμογή στις αυτο–αναφορικές δομές καθιστώντας τη C ιδιαίτερα ευέλικτη στη διαχείρισή τους. Μπορείτε να ανατρέξετε στην ενότητα 6.5 «Αυτο–αναφορικές Δομές» του [Kernighan '88] για να δείτε πώς χρησιμοποιούνται οι δείκτες σε αυτο–αναφορικές δομές δένδρων και στην ενότητα 17.5 του [King '96], όπου μπορείτε να δείτε πώς χρησιμοποιούνται οι δείκτες σε συνδεδεμένες λίστες (linked lists). Η διαχείριση αυτο–αναφορικών δομών και συνδεμένων λιστών ανήκουν στην κατηγορία των δύσκολων εφαρμογών που η κατανόησή τους αποδεικνύει καλή γνώση των εννοιών της γλώσσας C.

Άσκηση Αυτοαξιολόγησης 9.4

Τροποποιήστε τη συνάρτηση που ορίσατε στην άσκηση αυτοαξιολόγησης 3, ώστε να επιστρέφει `void`. Δώστε ένα παράδειγμα κλήσης της.

Δραστηριότητα 9.1

Αναπτύξτε ένα πρόγραμμα που θα χρησιμοποιεί τις τρεις διαφορετικές εκδόσεις της συνάρτησης `read_circle` που δώσατε στις ασκήσεις αυτοαξιολόγησης 3 και 4, για να παίρνει από τον χρήστη τις συντεταγμένες τριών κύκλων. Στη συνέχεια, να εμφανίζει τις συντεταγμένες τους στην οθόνη μαζί με την περίμετρο και την επιφάνειά τους. Τη δική μας έκδοση θα βρείτε στο τέλος του βιβλίου.

Άσκηση Αυτοαξιολόγησης 9.5

Για να θυμηθείτε λίγο την επίδραση του τελεστή μοναδιαίας αύξησης στους δείκτες σας δίνω τον παρακάτω κώδικα:

```
struct address addr[10];
struct address *addr_ptr = &addr[0];
   /* ή struct address *addr_ptr = addr; */
addr_ptr++;
```

Πού θα δείχνει ο δείκτης `addr_ptr`;

Σύνοψη

Η αφαιρετικότητα στα δεδομένα αποτελεί σε συνδυασμό με την αφαιρετικότητα στις διεργασίες ένα από τα ισχυρότερα όπλα του προγραμματιστή στην προσπάθειά του να δομήσει τη λύση κάθε σύνθετου προβλήματος. Ο ορισμός νέων τύπων, επιτρέπει την εύκολη αναπαράσταση και διαχείριση σύνθετων εννοιών του φυσικού προβλήματος. Η C, όπως και η Pascal, διαθέτει μηχανισμούς για την αναπαράσταση σύνθετων δεδομένων, τα οποία συνδυαζόμενα με τις κατάλληλες συναρτήσεις διαχείρισής τους, αποτελούν τους οριζόμενους από το χρήστη τύπους. Οι αντικειμενοστρεφείς γλώσσες διαθέτουν ένα πιο ισχυρό μηχανισμό, ο οποίος επεκτείνει τον μηχανισμό της δομής της C ή της εγγραφής της Pascal.

Η λέξη–κλειδί `struct` της C μάς δίνει τη δυνατότητα να ορίσουμε τους δικούς μας τύπους για την αναπαράσταση σύνθετων στοιχείων του προβλήματός μας. Η δομή λειτουργεί σαν συναθροιστικός τύπος, όπως και ο πίνακας, αλλά τα μέλη της είναι, συνήθως, διαφορετικού τύπου και έχουν το δικό τους όνομα με το οποίο μπορούμε να αναφερόμαστε σε αυτά. Οι δομές συνδυαζόμενες με τους πίνακες και τους δείκτες αποτελούν από τα πλέον ισχυρά στοιχεία της

C, για την ανάπτυξη ευκολοσυντήρητων, επεκτάσιμων και αποδοτικών προγραμμάτων.

Βιβλιογραφία κεφαλαίου

[Kernighan '88]

Η δεύτερη έκδοση του βιβλίου «*The C Programming Language*» κυκλοφόρησε το 1988. καλύπτοντας πλέον την ANSI C. Ελληνική έκδοση σε μετάφραση από τον Κλειδάριθμο 1990.

[King '96]

King N. K., «*C Programming: A modern approach*», W.W.Norton & Company, Inc.

ΚΕΦΑΛΑΙΟ 10

Η Γλώσσα Προγραμματισμού Pascal

Σκοπός

Σκοπός του κεφαλαίου είναι να εισάγει μια άλλη γλώσσα προστακτικού προγραμματισμού, την Pascal, και να δείξει τον μεγάλο βαθμό ομοιότητάς της με τη C.

Προσδοκώμενα Αποτελέσματα

Όταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- *αναφέρετε και περιγράψετε τα βασικά τμήματα ενός Pascal προγράμματος,*
- *περιγράψετε τη βασική διαφορά της Pascal από τη C όσον αφορά την αφαιρετικότητα στις διεργασίες,*
- *περιγράψετε τη διαφορά μεταξύ συνάρτησης και διαδικασίας,*
- *κάνετε περιορισμένης έκτασης τροποποιήσεις σε προγράμματα Pascal,*
- *αναφέρετε πέντε τουλάχιστον διαφορές της Pascal από τη C.*

Έννοιες κλειδιά

- *τμήμα δηλώσεων (declaration part)*
- *σώμα προγράμματος (program body)*
- *συνάρτηση (function)*
- *διαδικασία (procedure)*
- *Ada*
- *Modula*
- *Object Pascal*
- *repeat until*

Εισαγωγικές Παρατηρήσεις

Η Pascal είναι και αυτή μια διαδικαστική (procedural) γλώσσα προστακτικού προγραμματισμού. Οι έννοιες που υλοποιεί η Pascal αντιστοιχούν άμεσα σε αυτές της C με τη διαφορά ότι χρησιμοποιούνται διαφορετικές λέξεις για την έκφρασή τους. Γενικά η C θεωρείται δυσκολότερη, καθώς δίνει μεγαλύτερη ελευθερία στον προγραμματιστή. Το γεγονός αυτό για ένα μη πεπειραμένο προγραμματιστή δρα αρνητικά, αφού το περιβάλλον τού επιτρέπει να κάνει λάθη που δύσκολα ανιχνεύονται [Kernighan '81].

Το κεφάλαιο αυτό επιχειρεί μια σύντομη εισαγωγή στην Pascal δίνοντας τις

αντιστοιχίες με τη C για να σας βοηθήσει στην κατανόηση. Στόχος βέβαια δεν είναι να μάθετε Pascal αλλά να διαπιστώσετε πως οι βασικές έννοιες με τις οποίες ήρθατε σε επαφή στη θεματική αυτή ενότητα συναντώνται σχεδόν όλες και στην Pascal και, αν χρειαστεί, θα μπορέσετε πολύ σύντομα να εξοικειωθείτε με αυτή ή με κάποια από τις απογόνους της. Είναι προφανές ότι το κεφάλαιο υποθέτει καλή γνώση των αντίστοιχων εννοιών και του τρόπου με τον οποίο η C τις υλοποιεί. Θα σας συνιστούσα κατά τη διάρκεια της μελέτης του κεφαλαίου να ανατρέχετε στα αντίστοιχα προηγούμενα κεφάλαια του βιβλίου.

10.1 Γενικά

Η Pascal αναπτύχθηκε το 1971 από τον Niklaus Wirth. Αποτελεί γλώσσα υψηλού επιπέδου, ανήκει στην κατηγορία των γλωσσών προστακτικού προγραμματισμού και είναι, ίσως, η πιο δημοφιλής γλώσσα για τη διδασκαλία των βασικών εννοιών προγραμματισμού. Υποστηρίζει, όπως ήδη αναφέραμε, το δομημένο προγραμματισμό και αποτέλεσε τη βάση για τη δημιουργία άλλων γλωσσών όπως οι Ada, Modula, Object Pascal και UCSD Pascal. Ευρέα διαδεδομένο εξάλλου είναι το πανίσχυρο περιβάλλον ανάπτυξης εφαρμογών Delphi, που αποτελεί το διάδοχο της Turbo Pascal και υποστηρίζει το αντικειμενοστρεφές παράδειγμα. Όπως και η C, η Pascal διαθέτει το δικό της πρότυπο αναγνωρισμένο από την ANSI και το διεθνή οργανισμό τυποποίησης (ISO) που διασφαλίζει τη φορητότητα των προγραμμάτων Pascal.

Η Pascal υιοθετεί διαδικασία ανάπτυξης ανάλογη με αυτή της γλώσσας C. Η διαδικασία ξεκινά με τη φάση της ανάλυσης που περιλαμβάνει κατανόηση του προβλήματος. Ακολουθεί η φάση σχεδιασμού που περιλαμβάνει το σχεδιασμό του αλγόριθμου επίλυσης και, στη συνέχεια, ακολουθεί η φάση της υλοποίησης, η οποία περιλαμβάνει τη συγγραφή του αλγόριθμου με χρήση της Pascal. Πρώτο, μάλιστα, βήμα της φάσης της υλοποίησης είναι η ενημέρωση του μεταγλωττιστή της Pascal για τα δεδομένα του προβλήματος. Προσδιορίζονται τα ονόματα των κελιών μνήμης, που θα χρησιμοποιηθούν για αποθήκευση των δεδομένων, και οι τύποι των δεδομένων, που θα αποθηκευτούν σε κάθε κελί. Έτσι, ο μεταγλωττιστής γνωρίζει ποιες λειτουργίες είναι αποδεκτές για κάθε κελί μνήμης που χρησιμοποιείται από το πρόγραμμα.

10.2 Τύποι δεδομένων

Η Pascal διαθέτει τέσσερις ενσωματωμένους βασικούς τύπους : `Real` (για πραγματικούς αριθμούς), `Integer` (για ακεραίους), `Char` (για τον τύπο του χαρακτήρα) και τον `Boolean` (για τιμές αληθές/ψευδές). Μια τιμή μπορεί να εμφανίζεται στον κώδικα και ονομάζεται *κυριολεκτικό* (literal).

Εκτός από τους βασικούς τύπους, η Pascal υποστηρίζει, επιπλέον, τους συναθροιστικούς τύπους του πίνακα και της *εγγραφής* (`record`). Για τη δήλωση ενός πίνακα 10 ακεραίων έχουμε:

```
int a [10];                    /* C κώδικας */
a: array [0..9] of integer;    { Pascal κώδικας }
```

Η εγγραφή αντιστοιχεί με τη δομή της C, δίνοντας τη δυνατότητα εφαρμο-

γής της αφαιρετικότητας στα δεδομένα. Ο παρακάτω πίνακας 10.1 δίνει την ισοδυναμία.

Πίνακας 10.1

Εγγραφή της Pascal – δομή της C

C struct	Pascal record
`struct rec{`	`type`
`int a, b, c;`	`rec = record`
`float d, e, f;`	`a, b, c: integer;`
`};`	`d, e, f: real;`
	`end;`
`struct rec r;`	`var`
	`r: rec;`

Η αναφορά στα μέλη της εγγραφής γίνεται ακριβώς με τον ίδιο τρόπο που γίνεται στη C.

10.3 Μορφή προγράμματος Pascal

Ένα πρόγραμμα Pascal αποτελείται από δύο τμήματα: το τμήμα δηλώσεων (declaration part) και το σώμα του προγράμματος (program body).

Το τμήμα δηλώσεων περιλαμβάνει τις δηλώσεις των σταθερών και των μεταβλητών αλλά, όπως θα δούμε στη συνέχεια, και τους ορισμούς των συναρτήσεων (functions) και διαδικασιών (procedures).

Στον πηγαίο κώδικα του σχήματος 10.1 μπορείτε να διακρίνετε τον τίτλο του προγράμματος (program heading), που προσδιορίζει το όνομα κάθε προγράμματος Pascal αλλά και τα κανάλια για είσοδο και έξοδο των δεδομένων του. Στη συγκεκριμένη περίπτωση, το πρόγραμμα CircleArea παίρνει την είσοδο από το πληκτρολόγιο και βγάζει την έξοδο στην οθόνη.

Το σώμα του προγράμματος, όπως βλέπετε, αρχίζει με τη δεσμευμένη λέξη `begin` περιλαμβάνει ένα σύνολο από προτάσεις και τελειώνει με τη δεσμευμένη λέξη `end`. Για τον σχηματισμό των προτάσεων χρησιμοποιούνται τελεστές παρόμοιοι με αυτούς της C (βλέπε ένθετο πλαίσιο).

Προσέξτε, επίσης, τις κλήσεις των διαδικασιών Writeln και Readln για είσοδο και έξοδο δεδομένων, αντίστοιχα με τις `printf` και `scanf` της C. Είναι ενδιαφέρον στο σημείο αυτό να παρατηρήσουμε τη διαφορά της Pascal από τη C στο θέμα της αφαιρετικότητας στις διεργασίες. Η Pascal διαθέτει δυο

```
Program CircleArea (Input, Output);
{υπολογίζει την επιφάνεια ενός κύκλου}

const
   Pi = 3.14;
var
   radious,            {είσοδος — ακτίνα σε εκατοστά}
   area: Real; {έξοδος — επιφάνεια σε τετραγωνικά εκατοστά}

begin
   {διαβάζει την ακτίνα σε εκατοστά}
   Writeln ('Δώσε την ακτίνα σε εκατοστά>');
   Readln(Radious);

   {Υπολογίζει την επιφάνεια}
   Area := Pi * radious * radious;

   {Εμφανίζει το αποτέλεσμα}
   Writeln('Η επιφάνεια σε τετραγωνικά εκατοστά είναι ', Area)
end.
```

Σχήμα 10.1
Παράδειγμα Pascal προγράμματος.

μηχανισμούς για την εφαρμογή της αφαιρετικότητας στις διεργασίες: τη συνάρτηση και τη διαδικασία [Horowitz '95].

Τελεστές της Pascal

Ο πίνακας 10.2 δίνει ορισμένους από τους ευρέως χρησιμοποιούμενους τελεστές της C και τους ισοδύναμους της Pascal.

Πίνακας 10.2: *Αντιστοιχία τελεστών C και Pascal.*

C	Pascal	C	Pascal
=	:=	&&	and
==	=	||	or
!=	<>	!	not
%	mod	/ (για ακέραιους)	div

Παρενθέσεις, προτεραιότητα και προσεταιριστικότητα έχουν την ίδια σημασία για τον υπολογισμό εκφράσεων όπως και στη C.

Η παρατήρηση του Elliot Koffman «Μόνο το 25% του χρόνου που δαπανάται για ένα πρόγραμμα αφιερώνεται στον αρχικό σχεδιασμό και υλοποίηση, το υπόλοιπο 75% αφορά τη συντήρησή του» χρησιμοποιείται για να τονιστεί ακόμη μια φορά πως η χρήση κενών, κατάλληλων ονομάτων μεταβλητών συναρτήσεων και διαδικασιών, σχολίων, κ.λπ. βελτιώνουν την αναγνωσιμότητα του πηγαίου κώδικα και κάνουν τη συντήρησή του πιο εύκολη υπόθεση.

Άσκηση Αυτοαξιολόγησης 10.1

Σας δίνετε παραπλεύρως ο Pascal κώδικας ενός προγράμματος που υπολογίζει το παραγοντικό του αριθμού 6. Κάντε το πρόγραμμα πιο γενικό ώστε να δέχεται από το χρήστη τον αριθμό του οποίου το παραγοντικό θα υπολογίσει.

```
{ Program to find factorial of 6 }
program factorial;
const
   value=6;
var
   i, j: integer;
begin
   j:=1;
   for i:=1 to value do
        j:=j*i;
   writeln('The factorial of ',value,' is ',j);
end.
```

10.4 Αφαιρετικότητα στις διεργασίες

Η αύξηση του βαθμού επαναχρησιμοποίησης κώδικα αποτελεί πρωταρχικό στόχο για κάθε προγραμματιστή. Η αφαιρετικότητα στις διεργασίες αποτελεί το βασικότερο μηχανισμό για την επαναχρησιμοποίηση αυτή. Η Pascal διαθέτει τους μηχανισμούς της διαδικασίας και της συνάρτησης για εφαρμογή της αφαιρετικότητας στις διεργασίες. Η διαδικασία, όπως και η συνάρτηση, είναι μια ανεξάρτητη μονάδα (program unit or module), που περιλαμβάνει ένα σύνολο από προτάσεις που επιλύουν ένα συγκεκριμένο υπο–πρόβλημα. Η διαφορά τους έγκειται[1] στο ότι οι διαδικασίες μπορούν να επι-

[1] Η επιστροφή τιμών γίνεται όχι με τον μηχανισμό επιστροφής τιμής, που χρησιμοποιείται από την συνάρτηση, αλλά έμμεσα με την κατά αναφορά μεταβίβαση παραμέτρων.

στρέψουν περισσότερες από μία τιμές, ενώ η συνάρτηση επιστρέφει μια μόνο τιμή. Η δήλωση μιας διαδικασίας έχει την παρακάτω μορφή:

procedure *<όνομα–διαδικασίας> (τυπικές παράμετροι);*
τμήμα δηλώσεων
begin
    σώμα διαδικασίας
end*;*

Το τμήμα δηλώσεων είναι προαιρετικό και απαιτείται μόνο όταν η διαδικασία έχει τις δικές της μεταβλητές ή σταθερές.

Οι διαδικασίες όπως και οι συναρτήσεις ενός Pascal προγράμματος τοποθετούνται στο τμήμα δηλώσεων πριν από το σώμα του προγράμματος. Επιπλέον, η Pascal επιτρέπει δήλωση διαδικασιών μέσα σε άλλες διαδικασίες, κάτι που η C απαγορεύει. Το χαρακτηριστικό αυτό επιτρέπει τη δημιουργία ενός επιπλέον επιπέδου αφαιρετικότητας στις διεργασίες, βελτιώνοντας την άρθρωση του κώδικα.

Η δήλωση μιας συνάρτησης αρχίζει με τη λέξη–κλειδί `function` και έχει την παρακάτω μορφή:

function *<όνομα συνάρτησης>(τυπικές παράμετροι) : τύπος επιστρεφόμενης τιμής;*
    τμήμα δηλώσεων
begin
    σώμα συνάρτησης
end*;*

Θα πρέπει να παρατηρήσουμε την απουσία πρότασης ανάλογης της return της C. Η τιμή που επιστρέφει μια Pascal συνάρτηση είναι η τιμή που έχει αποδοθεί, πριν το τέλος του σώματος της συνάρτησης, στο όνομα της συνάρτησης που χρησιμοποιείται σαν τοπική μεταβλητή. Σαν παράδειγμα, δίνεται στο σχήμα 10.2 η συνάρτηση υπολογισμού μέσου όρου που δέχεται σαν παραμέτρους το άθροισμα των αριθμών και το πλήθος τους.

Ενδιαφέρον παρουσιάζει ο μηχανισμός επιστροφής τιμών από μια διαδικασία. Ο μηχανισμός χρησιμοποιεί τη λίστα τυπικών ορισμάτων. Κάθε τυπικό όρισμα μπορεί να είναι είτε *παράμετρος τιμής* (value parameter), οπότε χρησιμοποιείται μόνο για πέρασμα πληροφορίας στην διαδικασία, είτε *παράμετρος μεταβλητής* (variable parameter), οπότε χρησιμοποιείται για επιστροφή απο-

τελέσματος αλλάζοντας την τιμή της αντίστοιχης πραγματικής παραμέτρου. Για να διακριθούν οι τυπικές παράμετροι για επιστροφή από τις αντίστοιχες για είσοδο χρησιμοποιείται η λέξη–κλειδί *var* πριν από τα ονόματά τους.

```
function ComputeAve        (NumItems: Integer;
                            Sum: Real) : Real;
begin
   if NumItems > 0 then
        ComputeAve := Sum / NumItems
   else
        ComputeAve := 0
end;
```

Σχήμα 10.2
Συνάρτηση ComputeAve.

Άσκηση Αυτοαξιολόγησης 10.2

Γράψτε μια διαδικασία που θα υπολογίζει το άθροισμα και το μέσο όρο δύο πραγματικών αριθμών.

Δραστηριότητα 10.1

Γράψτε μια συνάρτηση στη C που είναι ισοδύναμη με τη συνάρτηση που αναπτύξατε στην άσκηση αυτοαξιολόγησης 10.2. Στη συνέχεια, αναπτύξτε ένα πρόγραμμα που τη χρησιμοποιεί.

10.5 Διαμόρφωση της ροής εκτέλεσης προγράμματος

Η Pascal διαθέτει τις ίδιες σχεδόν προτάσεις με αυτές της C για τη διαμόρφωση της ροής εκτέλεσης του προγράμματος με λίγες διαφορές στη σύνταξη.

Η πρόταση `if` έχει τη μορφή

if *έκφραση* **then**
    πρόταση1
else
    πρόταση2

όπου οι προτάσεις μπορεί να είναι απλές ή σύνθετες, οπότε παρεμβάλλονται μεταξύ ενός `begin` και ενός `end`. Στην περίπτωση της ένθεσης προτάσεων `if`, ισχύει ο ίδιος κανόνας με αυτόν της C για τη συσχέτιση των `else` με τα `if`.

Για την επιλογή μιας από πολλές εναλλακτικές επιλογές, η Pascal διαθέτει την πρόταση `case` με την παρακάτω σύνταξη:

case *έκφραση–επιλογής* **of**
    ετικέτα$_1$: πρόταση$_1$;
    ετικέτα$_2$: πρόταση$_2$;
    …
    ετικέτα$_n$: πρόταση$_n$;
end

Η τιμή της *έκφρασης–επιλογής* υπολογίζεται και συγκρίνεται με κάθε ετικέτα, που είναι μια λίστα από μια ή περισσότερες πιθανές τιμές της *έκφρασης–επιλογής*. Μόνο μία πρόταση εκτελείται και ο έλεγχος μεταφέρεται στην πρόταση μετά το `end`.

Η πρόταση `while` έχει τη μορφή

while *έκφραση* **do**
    πρόταση

Σημαντική διαφοροποίηση παρατηρούμε στην πρόταση `for` που έχει τη μορφή

for *counter* := *initial* **to** *final* **do**
    πρόταση

Η πρόταση εκτελείται μια φορά για κάθε τιμή του απαριθμητή `counter` μεταξύ των τιμών `initial` και `final` συμπεριλαμβανομένων. Αντί για το `to` μπορεί να χρησιμοποιηθεί το `downto` εάν η αρχική τιμή είναι μεγαλύτερη της τελικής.

Όπως θα θυμόσαστε από το κεφάλαιο 7 (προτάσεις ελέγχου ροής), η Pascal διαθέτει και την πρόταση `repeat` . Η σύνταξη της είναι:

repeat
    σώμα πρότασης επανάληψης
until *έκφραση–τερματισμού*

Μετά από κάθε εκτέλεση του σώματος της επανάληψης η τιμή της *έκφρασης–τερματισμού* υπολογίζεται και η επανάληψη συνεχίζεται για ψευδή τιμή, τερματίζεται δε για αληθή.

Άσκηση Αυτοαξιολόγησης 10.3

Αναφέρατε τα κοινά στοιχεία και τις διαφορές μεταξύ της πρότασης της Pascal `repeat` και της `do–while` της C.

Σύνοψη

Στο κεφάλαιο αυτό κάναμε μια πολύ σύντομη αναφορά σε βασικά στοιχεία της γλώσσας προγραμματισμού Pascal για να δείξουμε πως οι γνώσεις που αποκτήσατε σας καθιστούν ικανούς να κατανοήσετε και χρησιμοποιήσετε σχετικά εύκολα και σύντομα μια άλλη γλώσσα προστακτικού προγραμματισμού. Αναφερθήκαμε σε βασικές έννοιες όπως η μορφή ενός Pascal προγράμματος, οι τύποι δεδομένων, οι δηλώσεις μεταβλητών και σταθερών, οι τελεστές και ο υπολογισμός εκφράσεων, οι προτάσεις ελέγχου ροής και η αντιμετώπιση της αφαιρετικότητας στις διεργασίες. Αν εξαιρέσει κανείς τη διάκριση συναρτήσεων από διαδικασίες, στα υπόλοιπα διαπιστώνουμε ελάχιστες διαφορές. Η Pascal χρησιμοποιεί τις αγκύλες για σχόλια, τα `begin` *και* `end` *για αρχή και τέλος προγράμματος αλλά και σύνθετης πρότασης, τις λέξεις–κλειδιά function και procedure για να δηλώσει πως ένα όνομα είναι συνάρτηση ή διαδικασία αντίστοιχα, τη λέξη–κλειδί* `var` *για να δηλώσει τυπικές παραμέτρους που χρησιμοποιούνται για επιστροφή τιμής.*

Βιβλιογραφία κεφαλαίου

[Kernighan ’81]

Kernighan W. Brian, «*Why Pascal is not my favorite programming language*» AT&T Bell Laboratories, Computer Science Technical Report No.100.

Μπορείτε να βρείτε ένα αντίγραφο του άρθρου σε ηλεκτρονική μορφή στη διεύθυνση http://netlib.bell–labs.com/cm/cs/cstr/100.ps.gz

Στο άρθρο υπάρχουν αρκετές αναφορές σε εργασίες αξιολόγησης και σύγκρισης των δύο γλωσσών C και Pascal.

[Koffman ’95]

Koffman B. Elliot «*Pascal*» 5th Edition, Addison–Wesley Publishing Company.

Το κλασικό βιβλίο της Pascal, απόλυτα σύμφωνο με το βασικό κορμό σειράς μαθημάτων της επιστήμης των υπολογιστών της ACM. Καλύπτει με επιτυχία, όχι μόνο τη γλώσσα Pascal, αλλά παρεμφερή θέματα, όπως problem solving και software engineering.

[Horowitz ’95]

Horowitz Ellis, «*Fundamentals of Programming Languages*», Third edition, Computer Science Press, 1995.

Απαντήσεις ασκήσεων αυτοαξιολόγησης

1.1

Οι βασικές υπολογιστικές διεργασίες που το σύστημα πρέπει να εκτελέσει είναι: α) η εισαγωγή στο σύστημα των μέσων ημερήσιων θερμοκρασιών,β) ο υπολογισμός της μέσης μηνιαίας θερμοκρασίας και γ) η εμφάνιση στο χρήστη της μέσης μηνιαίας θερμοκρασίας.

Αν δεν εντοπίσατε τις παραπάνω διεργασίες, μην ανησυχείτε, είναι νέες έννοιες και νέος τρόπος σκέψης, θα πρέπει όμως, να μελετήσετε πάλι την υποενότητα 1.1.4, ώστε να εξοικειωθείτε με την έννοια της διεργασίας.

Αν εντοπίσατε τις διεργασίες, μπράβο σας! Στη συνέχεια, θα δούμε πώς μπορούμε να τις περιγράψουμε, ώστε να μπορεί να τις κατανοεί και, άρα, εκτελεί ο υπολογιστής.

1.2

Η σωστή χρονική σειρά είναι:

1. συγγραφή πηγαίου κώδικα,
2. αποθήκευση πηγαίου κώδικα,
3. μεταγλώττιση,
4. αναφορά λαθών από μεταγλωττιστή,
5. σύνδεση,
6. αναφορά λαθών από συνδέτη,
7. εκτέλεση προγράμματος,
8. εμφάνιση αποτελεσμάτων προγράμματος.

Αν δεν τα καταφέρατε, μην απογοητεύεστε. Θα πρέπει όμως να διαβάσετε πάλι, πιο προσεκτικά αυτή τη φορά, την ενότητα 1.2 και μετά να εκτελέσετε τη δραστηριότητα 1. Η κατανόηση των παραπάνω διεργασιών είναι πολύ βασική για την παραπέρα μελέτη σας.

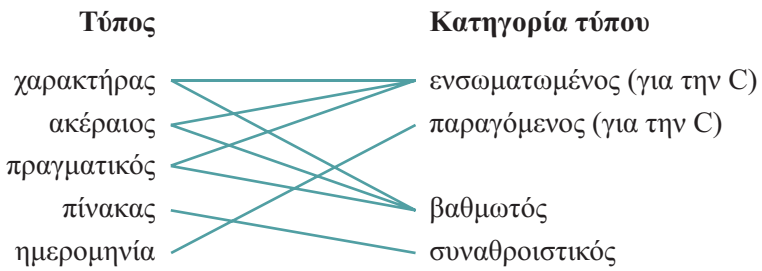
2.1

Σύμφωνα με τα μέχρι τώρα γνωστά από τα μαθηματικά, οι τελεστές +, –, *, /, < αναπαριστούν τις διεργασίες της πρόσθεσης, αφαίρεσης, πολλαπλασιασμού, διαίρεσης και σύγκρισης ανάμεσα σε αριθμούς αντίστοιχα.

2.2

Το συντακτικό μιας γλώσσας προγραμματισμού είναι ένα σύνολο κανόνων που καθορίζει αν μία πρόταση είναι σχηματισμένη σωστά ή όχι. Στη συνέχεια, θα δούμε ότι ένα πρόγραμμα αποτελείται από ένα σύνολο προτάσεων απλών ή σύνθετων.

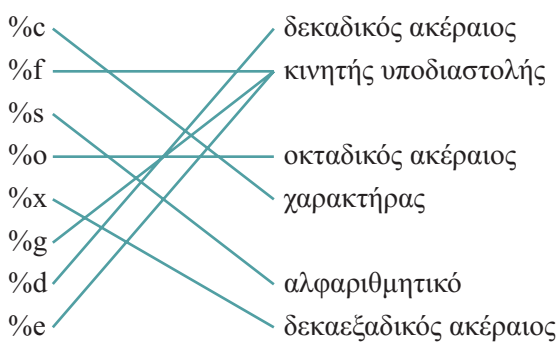
3.1



Μια ημερομηνία αποτελείται από την ημέρα, το μήνα και το έτος. Παρότι ορισμένες γλώσσες έχουν τον τύπο της ημερομηνίας (date) ενσωματωμένο, η C δεν τον έχει. Έτσι, ο προγραμματιστής πρέπει να παράγει τον τύπο `date` όπως έκανε και με τον τύπο φοιτητής.

Αν απαντήσατε σωστά, μπράβο σας. Αν όχι, μην απογοητεύεστε, αλλά πρέπει να ξαναδιαβάσετε, πιο προσεκτικά τώρα, την υποενότητα 3.1.3

3.2



Ο προσδιοριστής `%g` αφήνει τον υπολογιστή να επιλέξει για αριθμό κινητής υποδιαστολής την πιο σύντομη μορφή εμφάνισης.

Αν απαντήσατε σωστά, μπράβο σας, ήταν μια δύσκολη άσκηση που θέλει καλή μνήμη. Αν όχι, μην απογοητεύεστε, απλά θα πρέπει όταν παραστεί η ανάγκη να ανατρέξετε στο εγχειρίδιο με αποτέλεσμα να καθυστερήσετε στη συγγραφή ή κατανόηση του προγράμματος. Αρκεί, βέβαια, να ξέρετε που θα

ανατρέξετε (στην `printf`, θα έλεγε ένας προγραμματιστής).

4.1

Η εκτύπωση αλφαριθμητικής σταθεράς γίνεται με την printf χωρίς τη χρήση προσδιοριστή. Της περνάμε απλά την προς εκτύπωση αλφαριθμητική σταθερά.

```
printf("Hello");
```

Η εκτύπωση αλφαριθμητικού γίνεται με την `printf` χρησιμοποιώντας τον προσδιοριστή `s`. Η παρακάτω πρόταση

```
printf("Ο ISBN κωδικός του βιβλίου είναι : %s\n", isbn);
```

θα έχει σαν αποτέλεσμα να τυπωθεί στην οθόνη η πρόταση

Ο ISBN κωδικός του βιβλίου είναι 0–387–976–1

Αν οι προτάσεις σας δεν είναι όπως οι παραπάνω, πιθανότατα αυτό οφείλεται στο γεγονός ότι δεν έχετε πρόσβαση στην τεκμηρίωση της βασικής βιβλιοθήκης ή δεν έχετε ακόμη εξοικειωθεί με αυτήν. Επιμείνατε στο σημείο αυτό, είναι απαραίτητο για τη συνέχεια της μελέτης του κεφαλαίου.

Αν οι απαντήσεις σας είναι σωστές, μπράβο σας.

4.2

Αν δε θυμόσαστε τον προσδιοριστή που πρέπει να χρησιμοποιήσετε με την `scanf` δεν έχετε παρά να ανατρέξετε στην τεκμηρίωση της βασικής βιβλιοθήκης της C. Την κίνηση αυτή θα πρέπει να κάνετε πολύ συχνά από δω και πέρα.

Η εισαγωγή αλφαριθμητικού από την κύρια είσοδο γίνεται με την `scanf` και με τον ίδιο προσδιοριστή που χρησιμοποιεί και η `printf`. Η πρόταση

```
scanf("%s",isbn);
```

διαβάζει την κύρια είσοδο σαν αλφαριθμητικό και αποθηκεύει την τιμή στη μεταβλητή `isbn`.

Θα πρέπει όμως να τονίσουμε ότι:

α) Δεν χρειάζεται ο τελεστής `&` πριν από το όνομα της μεταβλητής `isbn`, όπως συνέβαινε για άλλους τύπους, όπως π.χ., του χαρακτήρα και του ακεραίου σε ανάλογη περίπτωση, γιατί το όνομα του πίνακα αναπαριστά την διεύθυνση του πρώτου στοιχείου του πίνακα και

β) η εισαγωγή περισσοτέρων από 11 χαρακτήρων έχει σαν αποτέλεσμα να καταστραφούν τα περιεχόμενα των επόμενων θέσεων μνήμης, που πιθανότατα αποτελούν τιμές άλλων μεταβλητών. Αυτό έχει απρόβλεπτα αποτελέσματα στην παραπέρα λειτουργία του προγράμματος. Ένα πολύ καλό παράδειγμα που τονίζει αυτό το τελευταίο πολύ σημαντικό πρόβλημα δίνεται στη σελίδα 141 του [King '96].

Αν δεν χρησιμοποιήσατε την `scanf` , θα πρέπει να ανατρέξετε στην ενότητα 3.2 για να δείτε πώς χρησιμοποιήσαμε την `scanf` για να εισάγουμε πληροφορία από την κύρια είσοδο.

Αν χρησιμοποιήσατε τον προσδιοριστή `&`, δεν πειράζει, θα πρέπει όμως από δω και μπρος να είστε πιο προσεκτικοί στις οδηγίες που σας δίνει η τεκμηρίωση της βασικής βιβλιοθήκης για τον τρόπο χρήσης των συναρτήσεών της.

Αν δώσατε τη σωστή απάντηση, μπράβο σας.

4.3

Για τη διαχείριση των δεδομένων του προβλήματός μας, μπορούμε να δηλώσουμε 40 ακέραιες μεταβλητές, αλλά είμαι σίγουρος ότι δεν θα συμφωνήσετε. Έχετε απόλυτο δίκαιο! Έχουμε ομοειδή τύπου δεδομένα και, άρα, πρέπει να χρησιμοποιήσουμε τον συναθροιστικό τύπο του πίνακα. Θα δηλώσουμε λοιπόν ένα πίνακα… πόσων στοιχείων 10, 4 ή 40; Οι δύο πρώτες επιλογές απορρίπτονται γιατί ο πίνακάς μας δεν θα μπορεί να δεχτεί τους βαθμούς όλων των μαθημάτων όλων των φοιτητών. Άρα, θα δηλώσω ένα πίνακα 40 στοιχείων.

```
int bathmos[40];
```

Πολύ ωραία! Τώρα μπορώ να αποθηκεύσω όλη την πληροφορία. Αλλά, ποιοι είναι οι βαθμοί του πρώτου φοιτητή; ποιοι είναι οι βαθμοί των φοιτητών για ένα μάθημα; Λίγο δύσκολο να απαντήσετε; Για δείτε τώρα την παρακάτω δήλωση πίνακα δύο διαστάσεων, σίγουρα θα σας αρέσει

```
int bathmos[10][4];
```

Η μεταβλητή `bathmos` είναι ένας πίνακας 10 στοιχείων (ένα για κάθε φοιτητή), όπου το κάθε στοιχείο του είναι πίνακας 4 ακεραίων (έναν για κάθε μάθημα του συγκεκριμένου φοιτητή). Αν και τα δεδομένα μας φαίνονται πολύ καλύτερα οργανωμένα τώρα, τα σημαντικότερα, αυτά που έχουν σχέση με την απλοποίηση της επεξεργασίας και, συνεπώς, έναν απλούστερο αλγό-

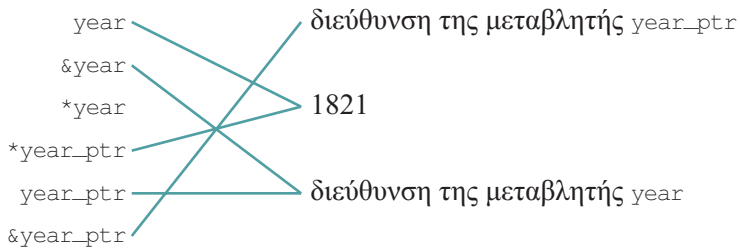
ριθμο, θα τα δούμε στη συνέχεια όταν θα αναφερθούμε στις προτάσεις ροής ελέγχου. Προς το παρόν, να πούμε πως η έκφραση `bathmos[0][1]` αναφέρεται στο βαθμό του δεύτερου ([1]) μαθήματος του πρώτου ([0]) φοιτητή, ενώ η αντίστοιχη `bathmos[1][0]` αναφέρεται στο βαθμό του πρώτου μαθήματος του δεύτερου φοιτητή.

4.4

Η σωστή απάντηση είναι `float temp[5][12][31];` και `temp[4][11][0]`

Αν απαντήσατε σωστά, μπράβο σας. Αν όχι, θα πρέπει να μελετήσετε πάλι την υποενότητα 4.1.6 και, κυρίως ,την απάντηση της άσκησης Αυτοαξιολόγησης 4.3

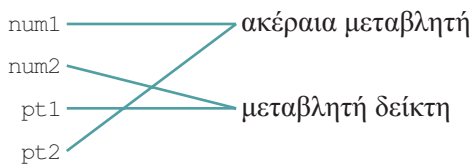
4.5



Ο τελεστής περιεχομένου έχει νόημα μόνο μπροστά από μεταβλητές δείκτη, άρα, η έκφραση `*year` δεν έχει νόημα. Αντίθετα, ο τελεστής `&` μπορεί να εφαρμόζεται και σε μεταβλητή δείκτη, οπότε και δίνει τη διεύθυνση στην οποία είναι αποθηκευμένος ο δείκτης.

Αν απαντήσατε σωστά συγχαρητήρια! Θα τα πάτε τέλεια με την C. Αν όχι, μην απελπίζεστε. Θα κατανοήσετε τις έννοιες με την πάροδο του χρόνου.

4.6



Ο τελεστής `*` στην πρόταση `int * pt1, pt2;` δεν αναφέρεται στο `int` αλλά στο `pt1`, δηλώνοντας ότι το περιεχόμενο του pt1 είναι ακέραιος (άρα το pt1 είναι δείκτης). Έτσι, το pt2 είναι ακέραια μεταβλητή και όχι δείκτης σε ακέραια μεταβλητή.

Αν αντιστοιχήσατε τα πάντα σωστά, τότε μπράβο σας. Αν όχι, μην απογοητεύεστε, το λάθος το κάνουν και καλοί προγραμματιστές.

4.7

Σωστή απάντηση είναι η Γ

Β. Σωστά κατά το ήμισυ. Η πρόταση ναι μεν δημιουργεί, όπως πολύ σωστά θέλουμε, τη διεύθυνση της `num (&num)` αλλά την αναθέτει (=) στην `*num_ptr`, η οποία είναι ακέραιος σύμφωνα με την `int *num_ptr`.

Γ. Μπράβο σας! Ο τελεστής & εφαρμόζεται πάνω στη μεταβλητή `num`, δίνει τη διεύθυνσή της και ο τελεστής = την αναθέτει στην `num_ptr`;

Α και Δ. Μετά τα παραπάνω σχόλια καταλαβαίνετε γιατί οι επιλογές αυτές είναι λάθος. Θα είναι καλύτερα όμως να ξαναμελετήσετε την ενότητα για τους δείκτες.

5.1

Το όνομα ενός πίνακα είναι έκφραση δείκτη.

Αν δεν απαντήσατε σωστά, δεν πειράζει, ήταν μια δύσκολη ερώτηση. Θα πρέπει όμως να επιμένετε περισσότερο στις ασκήσεις Αυτοαξιολόγησης. Προς το παρόν, θα πρέπει να ανατρέξετε στην Άσκηση Αυτοαξιολόγησης 2 για μια πιο επισταμένη μελέτη της.

Αν απαντήσατε σωστά, σας αξίζουν συγχαρητήρια. Είστε αρκετά παρατηρητικός.

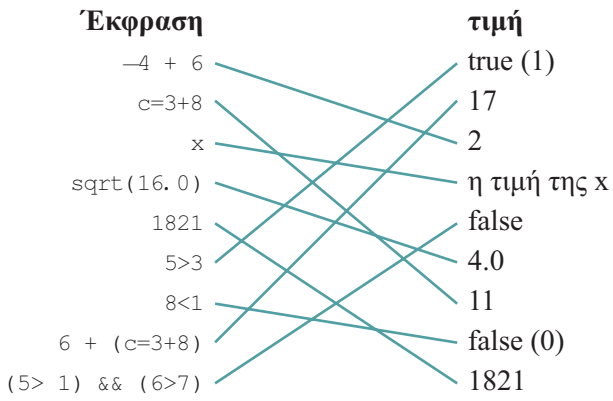
5.2

Πρώτα υπολογίζονται οι εσωτερικές παρενθέσεις. Οι εκφράσεις `(4 +2)` και `(7–5)` είναι στο ίδιο βάθος ένθεσης, ο υπολογισμός τους δίνει `2 + (6/2 –6)`. Στη συνέχεια, εφαρμόζεται ο τελεστής `/` και η έκφραση γίνεται `2 + (3 – 6)`. Εκτελείται η αφαίρεση και, τέλος, η πρόσθεση δίνει –1.

Αν δεν απαντήσατε σωστά, πρέπει να επιμείνετε στην υποενότητα 5.2.3 και ιδιαίτερα στις έννοιες της προτεραιότητας, της προσεταιριστικότητας και της επίδρασης των παρενθέσεων στον τρόπο υπολογισμού της τιμής της έκφρασης.

5.3

Η σωστή απάντηση ακολουθεί:



Όπως αναφέραμε, κάθε έκφραση έχει μία τιμή, η οποία προσδιορίζεται από την εκτέλεση πάνω στους τελεστέους, των διεργασιών που οι τελεστές της έκφρασης ορίζουν. Στις ειδικές περιπτώσεις περιλαμβάνονται:

Α) Έκφραση που δεν περιέχει τελεστή. Η τιμή της έκφρασης είναι η τιμή της μεταβλητής ή της σταθεράς, ή η τιμή που επιστρέφει η συνάρτηση της οποίας η κλήση αποτελεί την έκφραση. Στην κατηγορία αυτή ανήκουν οι παρακάτω εκφράσεις:

Έκφραση	τιμή
`x`	η τιμή της x
`1821`	1821
`sqrt(16.0)`	4.0
`12 + max(1,5)`	17

Η συνάρτηση `sqrt` υπολογίζει την τετραγωνική ρίζα, δέχεται όρισμα πραγματικό και επιστρέφει πραγματικό. Η `max` δέχεται δύο ορίσματα ακεραίους και επιστρέφει τον μεγαλύτερο.

Β) Έκφραση που περιέχει τον τελεστή ανάθεσης. Η τιμή της έκφρασης είναι η τιμή που ανατίθεται στον αριστερό τελεστέο της έκφρασης. Στην κατηγορία αυτή ανήκουν οι παρακάτω εκφράσεις:

Έκφραση	τιμή
`num = 21 — 12`	9
`num1 = num2 = 23`	23
`6 + (c=13+6)`	25

Γ) έκφραση που περιλαμβάνει συσχετιστικό τελεστή. Η τιμή μιας τέτοιας

έκφρασης μπορεί να είναι αληθής (true) ή ψευδής (false) και προσδιορίζει την αλήθεια ή όχι της συσχέτισης. Στην κατηγορία αυτή ανήκουν οι παρακάτω εκφράσεις:

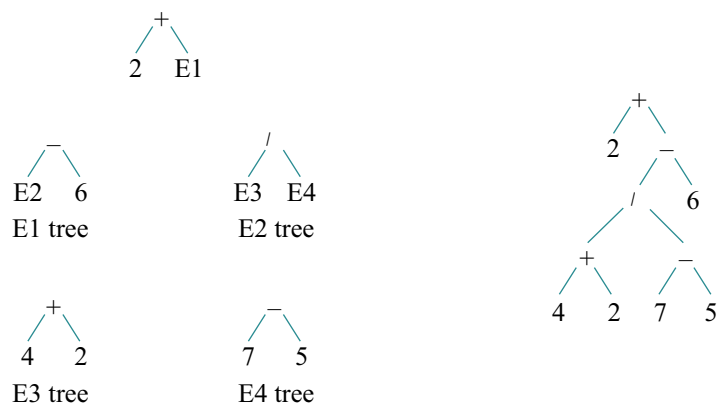
Έκφραση	τιμή
`5<6`	true
`5>6`	false (0)
`(5> 1) && (6>7)`	false
`(5> 1) || (6>7)`	true

Αν δεν απαντήσατε σωστά δεν πειράζει. Ορισμένες επιλογές ήταν πολύ δύσκολες για το επίπεδό σας. Μελετήστε όμως καλά την παραπάνω απάντηση. Αποτελεί προϋπόθεση για την παραπέρα μελέτη σας στις γλώσσες προγραμματισμού.

Αν απαντήσατε σωστά, σας αξίζουν συγχαρητήρια. Τα πάτε θαυμάσια.

5.4

Η έκφραση έχει την μορφή 2 + E1 όπου E1 είναι (E2–6), με E2 να είναι E3/E4 με τα E3 και E4 να είναι 4+2 και 7–5, αντίστοιχα.



Δημιουργούμε για κάθε υποέκφραση το αντίστοιχο δένδρο και, στη συνέχεια, ενοποιούμε τα δένδρα αντικαθιστώντας κάθε φύλλο που περιέχει έκφραση, με το αντίστοιχο δένδρο της έκφρασης. Παρατηρήστε ότι οι δύο υποεκφράσεις 4+2 και 7–5 είναι στο ίδιο επίπεδο του δένδρου. Στην περίπτωση αυτή, η σειρά υπολογισμού δεν είναι ορισμένη. Ένας μεταγλωττιστής μπορεί να υπολογίζει πρώτα την 4 + 2 ένας άλλος την 7 – 5. Υπάρχουν βέβαια ορισμένοι τελεστές για τους οποίους ορίζεται η σειρά υπολογισμού

των τελεστέων τους από αριστερά προς τα δεξιά.

Αν δεν απαντήσατε σωστά, δεν πειράζει, ήταν για σας η πρώτη κατασκευή δένδρου αφηρημένης σύνταξης. Ξαναδιαβάστε όμως την αντίστοιχη θεωρία ώστε να εξοικειωθείτε με τον μηχανισμό.

Αν κατασκευάσατε το σωστό δένδρο, μπράβο σας! Σύντομα θα έχετε καρπούς.

5.5

Η σωστή απάντηση είναι η Γ. `res1 = 2.56 + (float)3/2;`

Αν επιλέξατε την Α, χάσατε αρκετά. 3 δια 2 θα δώσει 1 σαν διαίρεση ακεραίων, συν το 2.56 μας κάμει 3.56. Όσο για τη μετατροπή; Είναι πολύ αργά.

Αν επιλέξατε την Β, πλησιάσατε αλλά πάλι χάσατε. 3 δια 2 θα δώσει 1 σαν διαίρεση ακεραίων. Το ένα και να το μετατρέψετε σε `float` πάλι 1 θα δώσει.

Αν επιλέξατε τη Γ, μπράβο σας. Αν, επιπλέον, σκεφτήκατε και όπως παρακάτω, τότε πετύχατε διάνα.

Η ρητή μετατροπή `(float)` μετατρέπει το 3 σε κινητής υποδιαστολής απλής ακρίβειας. Στη συνέχεια, το σύστημα για να διασφαλίσει στον τελεστή `/` ίδιου τύπου τελεστέους μετατρέπει αυτόματα το 2 σε τύπου `float`. Το αποτέλεσμα της εφαρμογής του τελεστή / θα είναι `float` . Στη συνέχεια, ο τελεστής `+` εφαρμοζόμενος σε δύο τελεστέους float δίνει αποτέλεσμα `float`, το οποίο ο τελεστής `=` αναθέτει χωρίς μετατροπή στη `float` μεταβλητή `res1`.

5.6

Η σωστή απάντηση είναι:

πρόταση	τιμή x	τιμή z
`z = ++x + y;`	11	31
`z = – –x + y;`	9	29
`z = x++ + y;`	11	30
`z = x– – + y;`	9	30

Είναι προφανές ότι στην περίπτωση του προπορευόμενου τελεστή το σύστημα πρώτα εκτελεί την αύξηση ή μείωση και μετά χρησιμοποιεί τη νέα τιμή της μεταβλητής στον υπολογισμό της τιμής της έκφρασης (προτάσεις α και

β). Αντίθετα, στην περίπτωση του παρελκόμενου τελεστή το σύστημα πρώτα χρησιμοποιεί την τιμή της μεταβλητής για τον υπολογισμό της τιμής της έκφρασης και μετά εκτελεί την αύξηση ή μείωση της τιμής της μεταβλητής (προτάσεις γ και δ).

Αν δεν απαντήσατε σωστά, μην ανησυχείτε, είναι ένας νέος τρόπος εφαρμογής, με τον οποίο όμως θα πρέπει να εξοικειωθείτε γιατί χρησιμοποιείται πολύ συχνά. Αναπτύξτε ένα μικρό πρόγραμμα με τις προτάσεις α – δ για να επιβεβαιώσετε τα αποτελέσματα με κατάλληλες κλήσεις της `printf` μετά από κάθε μια από τις παραπάνω προτάσεις. Πειραματιστείτε γράφοντας τις δικές σας εκφράσεις με χρήση των τελεστών `++` και `--`. Είναι ο καλύτερος τρόπος για να εξοικειωθείτε μαζί τους.

Αν απαντήσατε σωστά, σας αξίζουν συγχαρητήρια. Θα σας βοηθούσε και σας όμως η ανάπτυξη του παραπάνω προγράμματος.

5.7

```
int count = 10;
printf("%d %d\n",count,count*count++);
```

Το αποτέλεσμα μπορεί να είναι 10 100 ή 11 100 ανάλογα με το αν ο μεταγλωττιστής υπολογίζει πρώτα το πρώτο από αριστερά όρισμα ή το πρώτο από δεξιά, κάτι που δεν ορίζεται από την ANSI C, με αποτέλεσμα η πρόταση να δίνει διαφορετικά αποτελέσματα σε διαφορετικά συστήματα. Κατά ανάλογο τρόπο το αποτέλεσμα των προτάσεων

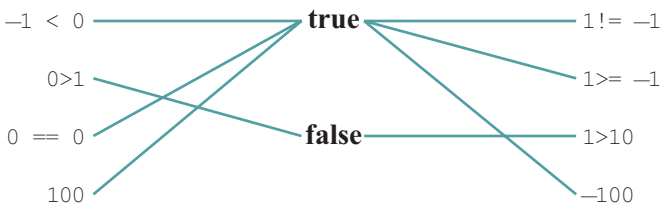
```
int num, count = 11;
num = count/2 + 5* (1+ count++);
```

μπορεί να είναι η ανάθεση στην `num` της τιμής 65 ή 66 ανάλογα αν θα υπολογιστεί πρώτα η έκφραση `count/2` ή η `1+count++`.

Μην ανησυχείτε που δεν βρήκατε τη σωστή λύση, είναι πραγματικά μια πολύ δύσκολη άσκηση και για έμπειρους C προγραμματιστές. Αν όμως μελετήσετε σωστά την υποενότητα 5.2.3, θα είστε σε θέση να αιτιολογήσετε και, άρα, να εντοπίσετε τέτοιου είδους προβλήματα στο μέλλον. Εξάλλου, ήταν μια άσκηση με σκοπό κυρίως να σας προβληματίσει για τις εκφράσεις αυτού του είδους.

Αν απαντήσατε σωστά, μπράβο σας! Θα ‘λεγα “μιλάτε άπταιστα C”.

5.8



Αν βρήκατε το στόχο σας, μπράβο σας! Συνεχίστε με τον ίδιο ρυθμό. Αν όμως αποτύχατε, θα πρέπει να επιμείνετε στην κατανόηση των συσχετιστικών τελεστών, αλλά και στον ορισμό του true και false. Είναι έννοιες που θα συναντάτε συχνά στη συνέχεια και δεν πρέπει να σας προβληματίζουν.

5.9

Η σωστή απάντηση είναι:

έκφραση	τιμή	έκφραση	τιμή
α) `count = 8`	8	γ) `num = count + 4`	12
β) `num == 9`	ψευδές	δ) `count == num – 4`	αληθές

Αν δεν απαντήσατε σωστά, μην ανησυχείτε, το λάθος αυτό είναι πολύ συνηθισμένο και όχι μόνο σε νέους προγραμματιστές. Προσέξτε όμως τώρα τη διαφορά που υπάρχει μεταξύ του τελεστή ανάθεσης (=) και του τελεστή ισότητας (==). Η χρήση του τελεστή ανάθεσης αντί του τελεστή ισότητας είναι σφάλμα που εντοπίζεται πολύ δύσκολα. Η έκφραση `num == 20` έχει τιμή 1 (αληθή) μόνο όταν η τιμή του `num` είναι 20. Αντίθετα, η `num = 10` έχει τιμή 10 και, άρα, αληθή για κάθε τιμή της num.

Αν απαντήσατε σωστά, σας αξίζουν πολλά, πολλά συγχαρητήρια, μπράβο σας!

5.10

Η σωστή απάντηση είναι:

	τιμή `x`	τιμή `y`	τιμή `z`	τιμή `(x++ < y--) && (y-- < ++z)`
α)	4	2	5	true
β)	6	3	4	false

Αν δεν απαντήσατε σωστά, μην ανησυχείτε, ήταν μια δύσκολη άσκηση. Θα πρέπει όμως να μελετήσετε πάλι πιο προσεκτικά την παράγραφο για τον υπολογισμό τιμής έκφρασης, καθώς και τους μοναδιαίους τελεστές αύξησης και μείωσης. Προσέξτε ότι στη δεύτερη περίπτωση, η συσχετιστική έκφραση

`(x++ < y--)` δίνει τιμή false και, σύμφωνα με τον υπολογισμό περιορισμένης έκτασης, το σύστημα δίνει αυτόματα τιμή false στη λογική έκφραση, χωρίς να υπολογίσει την έκφραση `(y-- < ++z)` και, άρα, δε μειώνει το `y` ούτε και αυξάνει το `z`. Το συμπέρασμα; Αποφεύγετε τη χρήση των τελεστών αυτών σε λογικές εκφράσεις.

Αν απαντήσατε σωστά, μπράβο σας! Έχετε εξοικειωθεί πολύ καλά με τη φιλοσοφία της C για τους τελεστές.

6.1

Η σωστή απάντηση είναι: `int max(int a, int b, int c);`

Αν η απάντησή σας δεν είναι σωστή, μην ανησυχείτε, είναι η πρώτη σας δήλωση. Θα πρέπει όμως για να τα πάτε καλύτερα στη συνέχεια, να μελετήσετε πάλι, πιο προσεκτικά αυτή τη φορά, την υποενεότητα 6.3.1.

Αν απαντήσατε σωστά, μπράβο σας, σε ένα επόμενο βήμα θα πρέπει να ολοκληρώσετε δίνοντας και τον ορισμό της συνάρτησης.

6.2

Η σωστή απάντηση είναι: `double power(double x, int n);`

Αν η απάντησή σας διαφέρει από την παραπάνω ως προς τη σειρά των τυπικών ορισμάτων, ή το όνομα της συνάρτησης, προσέξτε! Καλό είναι ο τρόπος αναφοράς σε μια συνάρτηση να πλησιάζει στο μέτρο του δυνατού τον τρόπο αναφοράς της διεργασία στο φυσικό πρόβλημα.

Αν δώσατε σωστή απάντηση, μπράβο σας, είναι πολύ σημαντικό να μπορείτε να δίνετε καλά πρωτότυπα συναρτήσεων. Αυξάνουν σε μεγάλο βαθμό την αναγνωσιμότητα του κώδικα.

6.3

Η σωστή απάντηση είναι

```
char *strchr(char *str,char ch);   ή
char *strchr(char str[],char ch);
```

Προσέξτε τον τρόπο με τον οποίο ορίζουμε σαν τυπικό όρισμα ένα αλφαριθμητικό. Είναι προφανές ότι το όρισμα αναπαριστά τη διεύθυνση του αλφαριθμητικού και όχι την τιμή του. Ένα εξίσου σημαντικό σημείο είναι η παράλειψη του μεγέθους του πίνακα `str`. Αν θέλουμε να περάσουμε την πλη-

ροφορία αυτή στη συνάρτηση θα πρέπει να χρησιμοποιήσουμε μια ακόμη παράμετρο, όπως για παράδειγμα την `int size`.

Αν δεν απαντήσατε σωστά, μην ανησυχείτε, ήταν μια δύσκολη άσκηση. Θα πρέπει όμως να φρεσκάρετε λίγο τα περί δεικτών στην Ενότητα 4.2.

Αν απαντήσατε σωστά, μπράβο σας, σας αξίζουν συγχαρητήρια.

6.4

Η σωστή απάντηση δίνεται σε δύο εκδόσεις με τη δεύτερη να δίνει πιο συμπαγή κώδικα και να εκμεταλλεύεται καλύτερα τις δυνατότητες της γλώσσας.

```
int max(int a, int b)
{
int max;
max = (a>b) ? a : b;
return(a);
}
```

α) πρώτη έκδοση

```
int max(int a, int b)
{
return((a>b) ? a : b);
}
```

β) Δεύτερη έκδοση

Αν η απάντησή σας δεν μοιάζει με τις παραπάνω, μην ανησυχείτε, είναι ο πρώτος ορισμός συνάρτησης, θα πρέπει όμως να μελετήσετε πιο προσεκτικά τα σχετικά με τη δήλωση και ορισμό συνάρτησης και να γυρίσετε για λίγο στο κεφάλαιο των τελεστών για να φρεσκάρετε τον υποθετικό τελεστή.

Αν απαντήσατε σωστά, μπράβο σας. Αν επιπλέον δώσατε και τη δεύτερη εκδοχή, πάτε θαυμάσια, θα σας ζήλευαν ακόμη και καλοί C προγραμματιστές.

6.5

Στη δήλωση των τυπικών ορισμάτων της συνάρτησης πρέπει να δίνουμε τον τύπο του κάθε ορίσματος, έστω και αν τα ορίσματα είναι ίδιου τύπου.

Η δήλωση της μεταβλητής ginomeno πρέπει να γίνει μέσα στο σώμα της συνάρτησης. Στο σημείο που έχει δοθεί δεν επιτρέπεται δήλωση μεταβλητής. Τέλος, λείπει το ; από την τελευταία πρόταση. Προσέξτε το σημασειολογικό λάθος ginomeno/base.

Αν απαντήσατε σωστά, μπράβο σας, είστε παρατηρητικοί, αν όχι θα πρέπει να μελετήσετε πάλι πιο προσεκτικά το κεφάλαιο 6 και να επιμείνετε στην κατανόηση των σφαλμάτων που σας αναφέρει ο μεταγλωττιστής σας.

7.1

Υπάρχει διαφορά μεταξύ των δύο προτάσεων και αυτή προσδιορίζεται στον αριθμό επαναλήψεων. Η Α χρησιμοποιεί την προθεματική σημειολογία, ενώ η Β τη μεταθεματική. Στην πρόταση Α αυξάνεται πρώτα η τιμή της count και η νέα τιμή της συγκρίνεται με το 12, ενώ στη Β πρώτα συγκρίνεται η τιμή της count με το 12 και, στη συνέχεια, αυξάνεται η τιμή της. Αυτό σημαίνει πως η πρόταση Π1 θα εκτελεστεί μία φορά παραπάνω στην περίπτωση Α.

Αν δεν απαντήσατε σωστά, σημαίνει πως δεν έχετε κατανοήσει καλά την έκδοση του παραδείγματος 4. Θα πρέπει να ανατρέξετε στην υποενότητα 5.5.1 περί τελεστών και, ειδικά στον τελεστή μοναδιαίας αύξησης, καθώς και στη θεωρία της πρότασης `while`.

Αν η απάντησή σας είναι σωστή, μπράβο σας, είστε παρατηρητικοί και προσέχετε τις λεπτομέρειες.

7.2

Ο βρόχος για τον υπολογισμό των μέσων ετήσιων θερμοκρασιών είναι

```
for(year=0; year<YEARS; year++){
   subtotal = 0;
   for(month=0; month<MONTHS; month++)
        subtotal+=temp[year][month];
   avg_temp[year] = subtotal/MONTHS;
   }
```

Ο βρόχος για τον υπολογισμό των μέσων μηνιαίων θερμοκρασιών είναι

```
for(month=0; month<MONTHS; month++){
   subtotal = 0;
   for(year=0; year<YEARS; year++)
        subtotal += temp[year][month];
   printf("Η μέση θερμοκρασία του %dου μήνα είναι: %.1f\n",
           subtotal/YEARS);
   }
```

Αν ο κώδικας δεν είναι όπως ο παραπάνω, δεν πειράζει, θα πρέπει όμως να μελετήσετε πάλι πιο προσεκτικά αφενός με τη θεωρία των πινάκων και μάλιστα των πολυδιάστατων και αφετέρου την πρόταση επανάληψης `for`. Οι ενθέσεις αυτής της μορφής δεν πρέπει να σας δημιουργούν πρόβλημα. Δοκι-

μάστε να αναπτύξετε το πλήρες πρόγραμμα της άσκησης Αυτοαξιολόγησης 3 ενθυμούμενοι τα σχόλια περί απλούστερου αλγόριθμου επεξεργασίας των στοιχείων.

Αν ο κώδικάς σας είναι όπως ο παραπάνω, τουλάχιστον όσον αφορά τις προτάσεις for, μπράβο σας, αν μάλιστα και τα ονόματα των μεταβλητών σας είναι παρεμφερή σάς αξίζουν πραγματικά συγχαρητήρια, πιάσατε το μήνυμα του «ευανάγνωστου κώδικα».

8.1

α) Η λέξη–κλειδί `static` στη δήλωση της καθολικής μεταβλητής `num` περιορίζει την ορατότητά της μόνο στο αρχείο που δηλώνεται. Αντίθετα, η `static` στη δήλωση της τοπικής μεταβλητής `count` δεν επηρεάζει την εμβέλειά της, αλλά ορίζει γι' αυτήν διάρκεια προγράμματος.

β) η `count` σαν τοπική μεταβλητή πλήρους διάρκειας αρχικοποιείται μία φορά μόνο κατά την είσοδο στο πρόγραμμα, αντίθετα με τη `num` που, επειδή δηλώνεται στην `func` σαν τοπική μεταβλητή περιορισμένης διάρκειας, αρχικοποιείται σε κάθε ενεργοποίηση της συνάρτησης `func`.

Αν απαντήσατε σωστά, μπράβο σας. Αν όμως η απάντησή σας δεν είναι η σωστή, θα πρέπει να δώσετε ιδιαίτερη σημασία στα θέματα της διάρκειας και εμβέλειας, είναι ζωτικής σημασίας για την οργάνωση των προγραμμάτων. Ξαναδιαβάστε πάλι πιο προσεκτικά τη μέχρι εδώ θεωρία του κεφαλαίου. Με κάθε τρόπο θα πρέπει να ξεκαθαρίσετε τις έννοιες αυτές.

8.2

Η σωστή απάντηση δίνεται παραπλεύρως. Καλό θα ήταν να ανατρέξτε στην ενότητα 7.1.1 του [Darnel '91] όπου μπορείτε να δείτε την αιτιολόγηση της απάντησης καθώς και μια πιο εκτενή αναφορά στις διαφορές που υπάρχουν στην αρχικοποίηση μεταβλητών πλήρους διάρκειας από αυτών περιορισμένης διάρκειας.

Αν δεν απαντήσατε σωστά, δεν πειράζει, η εφαρμογή των εννοιών αυτών θέλει ιδιαίτερη προσοχή. Μελετήστε πάλι τις υποενότητες 8.1.1 και 8.1.2 και δώστε έμφαση στην σε βάθος κατανόηση των κανόνων εμβέλειας.

```
j:2     k:2
j:2     k:3
j:2     k:4
```

8.3

Είναι προφανές ότι η συνάρτησή μας θα έχει σαν τυπικά ορίσματα δύο δείκτες τύπου int. Τα πραγματικά δε ορίσματα θα πρέπει να είναι οι διευθύνσεις των μεταβλητών οι οποίες αναπαριστούν τους ακέραιους αριθμούς μας. Έτσι, η σωστή απάντηση διαμορφώνεται όπως παραπλεύρως.

```
int max(int *,int *);
main()
{
:
max_value = max(&num1,&num2);
}

int max(int *a, int *b)
{
return(*a>*b?*a:*b);
}
```

Αν ο κώδικάς σας είναι σωστός, σας αξίζουν συγχαρητήρια, πάτε πολύ καλά. Αν, αντίθετα, δεν ορίσατε ή δεν καλέσατε σωστά τη συνάρτηση, θα πρέπει να επιμείνετε στο θέμα της μεταβίβασης με αναφορά, ανατρέχοντας ταυτόχρονα και στην ενότητα 4.2.

8.4

Η συνάρτηση θα ενεργοποιηθεί πέντε φορές. Σε κάθε ενεργοποίησή της, η τιμή της τυπικής παραμέτρου θα είναι κατά 1 μικρότερη της τιμής της τυπικής παραμέτρου της προηγούμενης ενεργοποίησης. Η επιστρεφόμενη τιμή θα είναι το άθροισμα της επιστρεφόμενης τιμής της χρονικά επόμενης ενεργοποίησης και της τιμής της τυπικής παραμέτρου. Η λογική αυτή μας οδηγεί στη συμπλήρωση του παρακάτω πίνακα:

Αριθμός ενεργοποίησης συνάρτησης	**τιμή τυπικής παραμέτρου n**	**επιστρεφόμενη τιμή sum(n–1) + n**
1	5	15
2	4	10
3	3	6
4	2	3
5	1	1

Αν δεν απαντήσατε σωστά, μην απελπίζεστε, είναι μια από τις δυσκολότερες ασκήσεις. Προϋποθέτει την σε βάθος κατανόηση των μηχανισμών της γλώσσας, κάτι που συμβαίνει με την πάροδο του χρόνου και τη συστηματική εξάσκηση με τον προγραμματισμό.

Αν απαντήσατε σωστά, σας αξίζουν πολλά, πολλά συγχαρητήρια. Θα έλεγα ότι «είστε γεννημένος προγραμματιστής».

9.1

Ο κύκλος μπορεί να αναπαρασταθεί από μία δομή με τρία μέλη κινητής υποδιαστολής. Σύμφωνα με αυτή τη θεώρηση, ο ορισμός της δομής θα είναι:

```
struct circle {
   float x, y;       /* συντεταγμένες κέντρου */
   float r;          /* ακτίνα */
   }
```

Για την αναπαράσταση των ζητούμενων κύκλων δηλώνουμε δύο μεταβλητές `struct circle` και αποδίδουμε αρχικές τιμές.

```
struct circle       c1 = {2.0, 3.0, 1.0},
                    c2 = {4.0, 6.0,  2.0};
```

Αν δεν απαντήσατε σωστά, επιμείνατε στη μελέτη των υπο–ενοτήτων 9.2-9.4. Η έννοια του τύπου δεδομένων είναι σημαντικής σημασίας για την επίλυση προβλημάτων με τον υπολογιστή.

9.2

Θεωρώντας τη διεύθυνση «Νίκος Νικολάου, Τέλου Άγρα 7, 24662 Πάτρα» έχουμε την παρακάτω δήλωση της μεταβλητής `addr1` με ταυτόχρονη απόδοση αρχικής τιμής.

```
struct address {      /* ορισμός κατάλληλης δομής */
   char name[40];
   char street[12];
   int number;
   int zip;
   char city[15];
   };
```

```
/* δήλωση μεταβλητής */
struct address addr1 = { "Νίκος Νικολάου", "Τέλου Αγρα", 7, 24662,
                         "Πάτρα"};
```

Θα πρέπει να τονίσουμε πως είναι ευθύνη του προγραμματιστή τα αλφαριθμητικά μέλη της δομής να είναι αρκετά μεγάλα ώστε να είναι ικανά να δεχτούν τις τιμές που αυτός τους αποδίδει.

Αν δεν απαντήσατε σωστά, επιμείνατε στη μελέτη των υπο–ενοτήτων 9.2–9.4. Θα διαπιστώσετε ότι, ο ορισμός, η δήλωση και η απόδοση αρχικών τιμών δεν ξεφεύγουν από τους κανόνες δήλωσης και απόδοσης αρχικών τιμών των βασικών τύπων της γλώσσας.

Αν απαντήσατε σωστά, μπράβο σας. Η χρήση δομών είναι αναγκαία σε κάθε σχεδόν εφαρμογή.

9.3

Ο κύκλος μπορεί να θεωρηθεί σαν συνάθροιση του κέντρου του, που είναι τύπου `point` και της ακτίνας του.

```
struct point {
   float x, y;          /* συντεταγμένες του σημείου */
   };

struct circle {
   struct point p;      /* το κέντρο του κύκλου */
   float r;             /* η ακτίνα του κύκλου */
   };
```

Η συνάρτηση θα δέχεται σαν όρισμα, μια δομή τύπου `circle`, θα παίρνει από το χρήστη τις τιμές και θα τις αποδίδει στα κατάλληλα πεδία της μεταβλητής. Μια έκδοση της συνάρτησης έχει ως κατωτέρω

```
struct circle read_circle(struct circle c1)
{
printf("Δώσε συντεταγμένες κέντρου: ");
scanf("%f %f", &c1.p.x, &c1.p.y);
printf("Δώσε την ακτίνα: ");
scanf("%f",&c1.r);
return c1;
}
```

Σαν παράδειγμα κλήσης μπορούμε να έχουμε

```
c = read_circle(c);
```

όπου `c` η μεταβλητή, στα μέλη της οποίας θέλουμε ο χρήστης να αποδώσει τιμές. Θα πρέπει να παρατηρήσουμε πως η συνάρτηση θα μπορούσε να μη δέχεται όρισμα, αλλά να δηλώνει μια τοπική μεταβλητή τύπου `circle` την οποία θα πρέπει να επιστρέφει.

```
struct circle read_circle(void)
{
struct circle c;
   /* κώδικας για την απόδοση τιμών στα μέλη της c */
return c;
}
```

Αν δεν προσθέσατε την πρόταση return, θα πρέπει να προσέξετε ότι η συνάρτηση δέχεται το όρισμα `c1` με τιμή, γεγονός που σημαίνει ότι δουλεύει σε αντίγραφο και όχι στο πραγματικό όρισμα c και, ως εκ τούτου, πρέπει να επιστρέψει τιμή η οποία πρέπει να αποδοθεί στη `c`.

Αν απαντήσατε σωστά, σας αξίζουν πολλά συγχαρητήρια.

9.4

Για να επιστρέφει η συνάρτηση `void`, θα πρέπει αυτή να έχει πρόσβαση στο πραγματικό όρισμα που δέχεται και όχι σε αντίγραφό του, ώστε να του αποδίδει άμεσα τις τιμές. Είναι προφανές ότι η συνάρτηση θα πρέπει να δέχεται τη διεύθυνση της μεταβλητής διαμέσου της οποίας θα έχει άμεση πρόσβαση στα μέλη της. Μια τέτοια έκδοση της συνάρτησης έχει ως κατωτέρω

```
void read_circle(struct circle *c1)
{
printf("Δώσε συντεταγμένες κέντρου: ");
scanf("%f %f", &c1->p.x, &c1->p.y);
printf("Δώσε την ακτίνα: ");
scanf("%f", &c1->p.r);
}
```

Σαν παράδειγμα κλήσης μπορούμε να έχουμε

```
read_circle(&c);
```

όπου η `c` είναι η μεταβλητή τύπου `circle`, στην οποία θέλουμε ο χρήστης να αποδώσει τιμές.

Αν δεν χρησιμοποιήσατε σωστά τις εκφράσεις της `scanf`, δεν πειράζει, είναι μια δύσκολη άσκηση που συνδυάζει γνώσεις από συναρτήσεις, τρόπους περάσματος ορισμάτων, δομές και προτεραιότητες τελεστών, τα οποία με την ευκαιρία καλό είναι να «φρεσκάρετε».

Αν απαντήσατε σωστά, σας αξίζουν πολλά συγχαρητήρια, είστε σε θέση να συνδυάζετε τις γνώσεις που πήρατε από την ΘΕ, για την αντιμετώπιση ενός προβλήματος.

9.5

Σύμφωνα με την αριθμητική δεικτών, ο δείκτης αυξανόμενος κατά ένα δείχνει, όχι στο επόμενο κελί μνήμης, αλλά στο επόμενο στοιχείο. Άρα, στην περίπτωσή μας, ο δείκτης θα δείχνει στο δεύτερο στοιχείο του πίνακα, δηλαδή στο `addr[1]`.

Αν δεν απαντήσατε σωστά, θα πρέπει να ανατρέξετε στην ενότητα 4.2.5 που διαπραγματεύεται τη σχέση των δεικτών με τους πίνακες.

10.1

Το πρόγραμμα δεν θα έχει τη value σαν σταθερά αλλά σαν μεταβλητή, η οποία θα παίρνει τιμή από το χρήστη με χρήση της `Readln`. Θεωρήστε τις παρακάτω δύο γραμμές κώδικα

```
write('Enter a value:');
Readln(value);
```

καθώς και τη δήλωση της μεταβλητής `value`.

Αν δεν απαντήσατε σωστά, δεν πειράζει, είναι το πρώτο σας πρόγραμμα Pascal. Διαβάστε όμως πάλι προσεκτικά την ενότητα 10.3.

Αν απαντήσατε σωστά, μπράβο σας. Απομένει να χρησιμοποιήσετε ένα μεταγλωττιστή της Pascal για να τρέξετε το πρόγραμμά σας.

10.2

Η διαδικασία θα έχει δύο ορίσματα τιμής και δύο ορίσματα μεταβλητής, η δε δήλωσή της θα έχει ως κατωτέρω:

```
procedure ComputeAve (Num1, Num2 {είσοδοι διαδικασίας} :Real;
                      var Sum, Average {έξοδοι διαδικασίας} : Real) ;
```

```
begin
   Sum := Num1 + Num2;
   Average := Sum / 2.0;
end;
```

Προσέξτε τη λέξη–κλειδί `var` πριν από τις τυπικές παραμέτρους που χρησιμοποιούνται για επιστροφή τιμής, καθώς και τα σχόλια που μπορούν να παρεμβάλλονται στον πηγαίο κώδικα.

Αν δεν απαντήσατε σωστά, εξαιρώντας τη λέξη–κλειδί `var`, θα πρέπει να μελετήσετε πάλι την ενότητα 10.3 και να θυμηθείτε τα περί τυπικών και πραγματικών ορισμάτων της C.

Αν απαντήσατε σωστά, μπράβο σας. Διαπιστώσατε ήδη ότι οι βασικές έννοιες είναι ίδιες με αυτές της C, απλά αλλάζει ο συμβολισμός.

10.3

Και οι δύο προτάσεις ανήκουν στην κατηγορία των βρόχων επανάληψης *συνθήκης–εξόδου* που διασφαλίζει την, για μια τουλάχιστον φορά, εκτέλεση του σώματος της επανάληψης. Η διαφορά τους έγκειται στο γεγονός ότι η `do–while` επαναλαμβάνεται για αληθή τιμή της έκφρασης–επανάληψης, ενώ η `repeat` για ψευδή.

Αν δεν απαντήσατε σωστά, θα πρέπει να είστε πιο προσεκτικοί στις λεπτομέρειες. Ανατρέξτε στο κεφάλαιο 7 για να μελετήσετε την εισαγωγή στις προτάσεις επανάληψης.

Αν απαντήσατε σωστά, σας αξίζουν συγχαρητήρια. Ο σωστός χειρισμός των προτάσεων ελέγχου είναι ένα σημαντικό στοιχείο για τη συγγραφή αξιόπιστου κώδικα.

Απαντήσεις Δραστηριοτήτων

1.1

Το αποτέλεσμα από την εκτέλεση του προγράμματος είναι η εμφάνιση, στην οθόνη του υπολογιστή σας, της παρακάτω γραμμής:

το άθροισμα του 10 με το 20 είναι: 30

Είναι προφανές ότι δεν αναπτύξαμε το πρόγραμμα για να υπολογίσουμε το παραπάνω άθροισμα. Αν δεν εκτελέσατε το πρόγραμμα, επιμείνετε έως ότου τα καταφέρετε. Είναι απαραίτητη προϋπόθεση για να προχωρήσετε στη μελέτη της ενότητας.

1.2

Τα περισσότερα προγράμματα εγκατάστασης μεταγλωττιστή δημιουργούν κάτω από το ευρετήριο στο οποίο εγκαθιστούν το μεταγλωττιστή τα υποευρετήρια: \BIN για τα αρχεία εκτελέσιμου κώδικα του μεταγλωττιστή, του συνδέτη κ.ο.κ., \LIB για τα αρχεία βιβλιοθηκών και \H για τα αρχεία επικεφαλίδας. Στο αρχείο math.h, που θα βρείτε στο ευρετήριο H, θα εντοπίσετε τα γνωστά σας, από τα μαθηματικά, ονόματα cos, sin, tan, τα οποία αντιστοιχούν στις διεργασίες υπολογισμού συνημίτονου, ημίτονου και εφαπτόμενης.

3.3

Η πρόταση

```
printf("Ο Νίκος είπε, \" το \\ ονομάζεται backslash.\"\n ");
```

θα εμφανίσει στην οθόνη

Ο Νίκος είπε, " το \ ονομάζεται backslash."

3.4

Μια έκδοση του προγράμματος μπορεί να έχει την παρακάτω μορφή:

```
#include <stdio.h>
main()
{
int num;
printf("Δώσε ένα αριθμό από 66 έως 90:");
scanf("%d",&num);
```

```
printf("χαρακτήρας πριν: %c \tASCII κωδικός :%d",num—1, num—1);
printf("χαρακτήρας: %c \tASCII κωδικός :%d",num, num);
printf("χαρακτήρας μετά: %c \tASCII κωδικός :%d",num+1, num+1);
}
```

Η εκτέλεση του προγράμματος, για αριθμό από τον χρήστη 66, έχει σαν έξοδο την παρακάτω:

χαρακτήρας πριν: A ASCII κωδικός : 65
χαρακτήρας: B ASCII κωδικός : 66
χαρακτήρας μετά: C ASCII κωδικός : 67

Αν η απάντησή σας δεν μοιάζει με την παραπάνω, μην ανησυχείτε, είναι από τα πρώτα προγράμματά σας. Θα πρέπει όμως να διαβάσετε πιο προσεκτικά την ενότητα 3.2.2 και να επιμένετε περισσότερο πριν ανατρέξετε στη δική μας λύση.

Αν τα καταφέρατε, μπράβο σας, κάνετε πολύ καλά βήματα.

3.5

Χρησιμοποιήστε το χαρακτήρα «`\t`» για να στοιχίσετε τα αποτελέσματα. Για τους αριθμούς 1 και 2, το πρόγραμμα θα τυπώσει:

οκταδική μορφή	δεκαδική μορφή	δεκαεξαδική μορφή
3	3	3

3.6

Εάν η μέγιστη τιμή ενός short ακεραίου για το σύστημά σας είναι 32767 και εκτελέστε το παρακάτω πρόγραμμα

```
/* toobig.c — Integer Overflow
#include <stdio.h>
main()
{
short i = 32767;
printf("%d \t%d \t%d\n",i,i+1,i+2);
}
```

θα έχετε σαν αποτέλεσμα την εμφάνιση στην οθόνη της παρακάτω γραμμής:

32767 –32768 –32767

Η μεταβλητή `count` ενεργεί όπως ακριβώς ο δείκτης χιλιομέτρων των αυτο-

κινήτων. Μόλις φτάσει στη μεγαλύτερη τιμή του αρχίζει πάλι από την αρχή. Το φαινόμενο αυτό είναι γνωστό με το όνομα υπερχείλιση ακεραίου (integer overflow).

4.1

Παρακάτω δίνεται μια έκδοση του ζητούμενου προγράμματος.

Μια εναλλακτική μορφή θα μπορούσε να χρησιμοποιήσει μια μόνο κλήση της `printf`

```
printf("%s\n%c\n%c\n%c\n%c\n", name,name[1],name[2],name[3],name[9]);
```

```
#include <stdio.h>
char name[20] = "ΔΗΜΟΣΘΕΝΗΣ";
main()
{
printf("%s\n",name);
printf("%c\n%c\n%c\n",name[1],name[2],name[3]);
printf("%c\n",name[9]);
}
```

4.2

Προσέξτε τη συμπερίληψη του αρχείου επικεφαλίδας `string.h`. Είναι απαραίτητη για τη χρήση της συνάρτησης `strlen`. Η strlen δέχεται σαν παράμετρο δείκτη σε αλφαριθμητικό και επιστρέφει το μήκος του σε bytes χωρίς να συμπεριλαμβάνει το χαρακτήρα τερματισμού `'\0'`.

```
#include <stdio.h>
#include <string.h>
char name[20];
main()
{
printf("Δώσε όνομα: ");
scanf("%s",name);
printf("%s\n",name);
printf("%c\n%c\n%c\n",name[1],name[2],name[3]);
printf("%c\n",name[strlen(name)-1]);
}
```

5.1

Εάν `num1` και `num 2` οι δύο μεταβλητές στις οποίες έδωσε τιμές ο χρήστης, τότε οι προτάσεις

```
max = num1>num2 ? num1 : num2;
printf("ο μεγαλύτερος είναι ο %d\n", max);
```

ή η πιο συμπαγής πρόταση

```
printf("ο μεγαλύτερος είναι ο %d\n", num1>num2 ? num1 : num2);
```

εμφανίζει τον `num1` εάν `num1>num2` αληθές, διαφορετικά εμφανίζει το `num2`.

Αντίστοιχα, για τον μικρότερο έχουμε την πρόταση

```
printf("ο μικρότερος είναι ο %d\n", num1<num2 ? num1 : num2);
```

Για τους τρεις αριθμούς έχουμε τις προτάσεις

```
max1 = num1>num2 ? num1 : num2;
max2 = max1 > num3 ? max1 : num3;
printf(ο μεγαλύτερος είναι ο %d\n",max2);
```

Οι δύο πρώτες προτάσεις μπορούν να συμπτυχθούν εάν στη δεύτερη και στη θέση του `max1` βάλω το αριστερό μέλος της πρώτης πρότασης. Έτσι, έχω την πιο συμπαγή πρόταση

```
num1>num2 ? (num1>num3? : num1 : num3):
(num2>num3 ? num2 : num3);
```

Προσπαθήστε να κατανοήσετε και να εξοικειωθείτε με τις παραπάνω παραλλαγές. Αν σας φαίνεται δύσκολο, δεν έχετε άδικο. Θέλει πολύ προσεκτική εφαρμογή των εννοιών της προτεραιότητας και προσεταιριστικότητας των τελεστών, χωρίς αυτό να αποκλείει αναφορά στον αντίστοιχο πίνακα.

6.1

Είναι σημαντικό πριν από την κλήση κάθε συνάρτησης να υπάρχει στον πηγαίο κώδικα το πρωτότυπό της. Αυτό επιτρέπει στο μεταγλωττιστή να ελέγξει κάθε πρόταση κλήσης αν είναι σύμφωνη ως προς τον αριθμό και τον τύπο των τυπικών ορισμάτων, καθώς και τον τύπο της επιστρεφόμενης τιμής. Το πλαίσιο 9–1 του [Darnell 1991] κάνει μια καλή αναφορά στο πρωτότυπο συνάρτησης.

```
int max(int a, int b, int c);
double power(double x, int n);
char *strchr(char *str,char ch);
int num = 5;
char *ch_ptr;
main()
{
printf("%d\n",max(12/2,num+3,2*num));
printf("%f",power(num,3));
ch_prt = strchr("klea",'e');
}
```

6.2

Για τη χρήση των συναρτήσεων πρέπει να συμπεριληφθεί το αρχείο επικεφαλίδας `string.h`.

Η `strcpy` δέχεται σαν ορίσματα δύο αλφαριθμητικά, πιο συγκεκριμένα δύο δείκτες σε αλφαριθμητικά και αντιγράφει το δεύτερο αλφαριθμητικό στο πρώτο. Προσέξτε! Είναι ευθύνη του προγραμματιστή να διασφαλίσει ότι το αλφαριθμητικό αποδέκτης έχει αρκετό χώρο ώστε να «χωρέσει» το νέο περιεχόμενο. Ένα συχνό λάθος είναι η κλήση της συνάρτησης με πρώτο όρισμα ένα δείκτη σε χαρακτήρα, ο οποίος δεν έχει ενημερωθεί να δείχνει σε συγκεκριμένο χώρο της μνήμης. Κλήσεις της μορφής `strcpy(ch_ptr, str1)` αποτελούν ένα από τα πλέον δύσκολα εντοπιζόμενα σφάλματα στη C.

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

char str1[] = "Klea";
char str2[] = "nthis";
char str3[] = "klea";

char ch = 'a';
char bigstr[50];

main()
{
char *ch_ptr;
```

```
printf("bigstr: %s\n",bigstr);
strcpy(bigstr,str1);
printf("bigstr: %s\n",bigstr);
strcat(bigstr,str2);
printf("bigstr: %s\n",bigstr);
printf("str1 %s str2\n", strcmp(str1,str3)? "Not equal": "equal");
printf("str1 %s \"Klea\"\n", strcmp(str1,"Klea")?
       "Not equal": "equal");
ch_ptr = strchr(str1,ch);
printf("%c\n",ch_ptr?*ch_ptr: '#');
ch_ptr = strchr(str2,ch);
printf("%c\n",ch_ptr?*ch_ptr: '#');
}
```

Το πρόβλημα; Ναι μεν ο `ch_ptr` είναι δείκτης σε χαρακτήρα, όπως απαιτεί το πρωτότυπο της `strcpy`, αλλά δεν δείχνει σε θέση μνήμης που έχει στη διάθεσή του για χρήση το πρόγραμμά μας. Η κλήση θα ήταν σωστή, αν είχε προηγηθεί η πρόταση `ch_str = bigstr;` , η οποία βάζει το δείκτη `ch_str` να δείχνει στη μνήμη των 50 bytes που έχει δεσμεύσει το πρόγραμμα για τον πίνακα χαρακτήρων `bigstr`.

Προσέξτε, επίσης, ότι η επιστρεφόμενη τιμή της `strchr` είναι δείκτης σε χαρακτήρα. Δείχνει την πρώτη εμφάνιση του ζητούμενου χαρακτήρα μέσα στο αλφαριθμητικό. Άρα, μπορώ να αναφερθώ στο περιεχόμενό του εφαρμόζοντας πάνω στο δείκτη τον τελεστή περιεχομένου `*`. Η έκφραση είναι τύπου χαρακτήρα και τυπώνω την τιμή της χρησιμοποιώντας τον προσδιοριστή `%c` της `printf`.

6.3

Για τη χρήση των συναρτήσεων πρέπει να συμπεριληφθεί το αρχείο επικεφαλίδας `ctype.h`.

Οι συναρτήσεις `isalnum()`, `isalpha()`, `isdigit()`, `islower()`, και `isspace()` δέχονται σαν όρισμα μια ακέραια τιμή, για την οποία ελέγχουν αν ικανοποιεί την από το όνομα της συνάρτησης περιγραφόμενη ιδιότητα, οπότε και επιστρέφουν true αλλιώς επιστρέφουν false (0).

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

```
main()
{
int ch;

printf("%c \t%3s\n",65,isalnum(65)?"AN":"");
printf("%c \t%3s\n",222,isalpha(222)?"A":"");
printf("%c \t%3s\n",55,isdigit(55)?"D":"");
printf("%c \t%3s\n",92,islower(92)?"L":"");
printf("%c \t%3s\n",32,isspace(32)?"Space":"");
printf("%c \t %c\n",'A',tolower('A'));
printf("%c\t%c\t%c\n",'b', toupper('b'),toupper(tolower('B')));
}
```

Οι `tolower()` και `toupper()` μετατρέπουν το όρισμα που δέχονται σε μικρό χαρακτήρα ή μεγάλο αντίστοιχα, με την προϋπόθεση βέβαια ότι αυτός υπάρχει.

6.4

Για τη χρήση των συναρτήσεων πρέπει να συμπεριληφθεί το αρχείο επικεφαλίδας `stdio.h`.

Αν το πρόγραμμα σας δεν μοιάζει με το παραπλεύρως θα πρέπει να επιμείνετε στην ενότητα αυτή. Ασκήσεις αυτής της μορφής θα πρέπει να τις αναπτύσσετε χωρίς δυσκολία. Μελετήστε την ενότητα 6.3 αλλά και τις υποενότητες 4.1.4 και 4.1.5 που αναφέρονται στα αλφαριθμητικά.

```
#include <stdio.h>

char ch;
char line[80];

main()
{
printf("Δώσε μια πρόταση:");
gets(line);
printf("Η πρόταση που έδωσες είναι:");
puts(line);
printf("Δώσε χαρακτήρα:");
ch=getchar();
```

```
printf("Ο χαρακτήρας που έδωσες είναι:");
putchar(ch);
putchar('\n');
}
```

Θα διαπιστώσετε ότι σε συνδυασμό με την τεκμηρίωση της βασικής βιβλιοθήκης έχετε τα απαραίτητα εφόδια.

7.1

Αν αναπαραστήσουμε το έτος με την ακέραια μεταβλητή `year`, θα έχουμε:

```
if((year % 400) == 0) then το έτος είναι δίσεκτο
else if ((year % 100) == 0) then το έτος δεν είναι δίσεκτο
else if (year % 4) == 0 then το έτος είναι δίσεκτο
else το έτος δεν είναι δίσεκτο
```

Αν η απάντησή σας δεν περιέλαβε τους απαραίτητους ελέγχους, ίσως έπρεπε να σιγουρευτείτε πρώτα πότε ένα έτος είναι δίσεκτο. Προσέξτε, η γλώσσα προγραμματισμού, δεν σας βοηθά να λύσετε το πρόβλημα, αλλά μόνο αν έχετε βρει τον τρόπο που μπορείτε να φθάσετε στην λύση, σας επιτρέπει να περιγράψετε τα βήματα που πρέπει να ακολουθήσει ο υπολογιστής για να οδηγηθεί σε αυτή.

Αν η απάντησή σας είναι ίδια με την παραπάνω ή διαφέρει από αυτήν ως προς τη σειρά των προτάσεων, μπράβο σας, δομείτε πολύ σωστά τις προτάσεις σας. Αυτό θα σας βοηθήσει πολύ στον προγραμματισμό.

7.2

Σαν απάντηση σας δίνω τρεις εκδόσεις τις οποίες σχολιάζω

```
int max(int a, int b, int c)
{
if(a>b)
if(a>c)
return a;
else
return c;
else
if(b>c)
return b;
else
return c;
}
```

α) πρώτη έκδοση της max

```
int max(int a, int b, int c)
{
if(a>b)
   if(a>c)
      return a;
   else
      return c;
else
   if(b>c)
      return b;
   else
      return c;
}
```

β) δεύτερη έκδοση της max

```
int max(int a, int b, int c)
{
if(a>b)
   if(a>c)
      return a;
   else
      return c;
else if(b>c)
      return b;
   else
      return c;
}
```

γ) τρίτη έκδοση της max

Αλήθεια, πώς σας φαίνεται η πρώτη έκδοση; Υποθέτω δεν σας αρέσει καθόλου, όπως δεν θα άρεσε σε κάθε προγραμματιστή που θα έπρεπε να διαβάσει τον κώδικα και να καταλάβει τι κάνει ή να ελέγξει αν κάνει σωστά τη δουλειά του. Συγκρίνατε τώρα, την πρώτη έκδοση με τη δεύτερη ή τρίτη. Τα σχόλια περιττεύουν. Το συμπέρασμα; Δώστε την κατάλληλη σημασία στη σωστή στηλοθέτηση του κώδικά σας, είναι εξίσου σημαντικό όσο και το αν δουλεύει σωστά το πρόγραμμά σας. Προσέξτε επιπλέον τα παρακάτω:

Η πρόταση που είναι κάτω από το πρώτο if είναι πρόταση διακλάδωσης. Δεν είναι σύνθετη πρόταση έστω και αν αποτελείται από περισσότερες της μιας γραμμές κώδικα, και για το λόγο αυτό δεν είναι απαραίτητη η έγκλιση των γραμμών της σε αγκύλες. Το ίδιο συμβαίνει και για την πρόταση που ακολουθεί το `else` του πρώτου `if`. Τώρα, βέβαια, θα ρωτήσετε «τι σημαίνει `else` του πρώτου `if`»;

Ο μεταγλωττιστής συσχετίζει τα `else` με τα `if` σύμφωνα με τον ακόλουθο κανόνα: «Ένα `else` συσχετίζεται πάντα με το πλησιέστερο προηγούμενο `if`» και αυτό, βέβαια, ανεξάρτητα από τη στηλοθέτιση του κώδικα[1]. Μια καλή στηλοθέτιση όμως, βοηθά τον αναγνώστη του κώδικα να κάνει γρήγορα τις συσχετίσεις και να κατανοήσει τη δομή του κώδικα. Ένα `else` πρέπει να είναι πάντα στον ίδιο στηλοθέτη με το `if` στο οποίο αντιστοιχεί.

[1] Αυτός είναι ο λόγος που και η πρώτη έκδοση λειτουργεί σωστά.

Όσον αφορά τον υποθετικό τελεστή ; ναι, μπορεί να χρησιμοποιηθεί. Αν δεν συμφωνείτε ανατρέξτε στη Δραστηριότητα 4.

Αν η απάντησή σας δεν είναι η σωστή, μην ανησυχείτε, ήταν μια δύσκολη άσκηση. Διαβάστε όμως πάλι προσεκτικά την αντίστοιχη θεωρία και σιγουρευτείτε πως κατανοήσατε πλήρως τα παραπάνω σχόλια.

Αν φτιάξατε τη συνάρτηση, σας αξίζουν συγχαρητήρια. Αν μάλιστα δώσατε και την έκδοση με τον τριαδικό τελεστή, μου φαίνεται πως θα εξελιχθείτε σε πολύ καλό C προγραμματιστή.

7.3

Μια έκδοση του ζητούμενου προγράμματος έχει ως κατωτέρω

```
#include <stdio.h>

void main()
{
float num1,num2;
float result;
int choice;

printf("Δώσε τον πρώτο αριθμό: ");
scanf("%f",&num1);
printf("Δώσε τον δεύτερο αριθμό: ");
scanf("%f",&num2);

printf("\n\n");
printf("\n 1 – Πρόσθεση \n");
printf("\n 2 – Αφαίρεση \n");
printf("\n 3 – Πολλαπλασιασμός \n");
printf("\n 4 – Διαίρεση \n");
printf("Επιλέξατε από 1–4: ");
scanf("%d",&choice);
```

(συνέχεια…)

(… συνέχεια)

```
   switch(choice) {
   case 1:
        result = num1 + num2;
        break;
   case 2:
        result = num1 – num2;
        break;
   case 3:
        result = num1 * num2;
        break;
   case 4:
        if(num2)
               result = num1 / num2;
        else
               printf("Δεν επιτρέπεται διαί-
                      ρεση με το μηδέν\n");
        break;
   default:
        printf("Η επιλογή δεν υποστηρίζεται \n");
   }
printf("Το αποτέλεσμα είναι %f, %f", result, result);
exit(0);
}
```

Παρατηρήστε πως η πρόταση με την εμφάνιση του αποτελέσματος εκτελείται και όταν ακόμη εμφανίζεται το μήνυμα «η επιλογή δεν υποστηρίζεται».

Η παρακάτω πρόταση

```
if(choice>=1 && choice=<4)
   printf("Το αποτέλεσμα είναι %f, %f", result, result);
```

τυπώνει το αποτέλεσμα μόνο αν επιλέχθηκε ένα από τα 1, 2, 3 ή 4. Αλλά, προσέξτε και όταν ακόμη αναγνωρισθεί η δια του μηδενός διαίρεση.

Τροποποιήστε τον κώδικα χρησιμοποιώντας τη συνάρτηση `display_menu` ή σε ένα επόμενο βήμα την `menu()`, η οποία, εκτός από την εμφάνιση των επιλογών που έχει ο χρήστης, δέχεται και την επιλογή του χρήστη την οποία επιστρέφει όπως στην παρακάτω πρόταση `choice = menu();`

Αν το πρόγραμμά σας δεν μοιάζει με το παραπάνω, θα χρειαστεί να επανέλθετε στη θεωρία τη σχετική με την πρόταση `switch` και στο παράδειγμα 3. Στη συνέχεια, προσπαθήστε να γράψετε από την αρχή τον κώδικα του προγράμματος. Θα δείτε ότι τελικά θα τα καταφέρετε.

Αν αναπτύξατε τον κατάλληλο κώδικα, μπράβο σας. Αξίζει όμως τον κόπο να ασχοληθείτε με τις τροποποιήσεις που σας πρότεινα παραπάνω.

7.4

Όπως θα θυμόσαστε, η εντολή `#define` του προεπεξεργαστή, σας δίνει τη δυνατότητα να συσχετίσετε ένα όνομα με μια σταθερά ώστε, στη συνέχεια, να αναφέρεστε στη σταθερά με το όνομά της. Χρησιμοποιώντας την `define` έχω τις παρακάτω δηλώσεις:

```
#define ADD          1
#define SUB          2
#define MUL          3
#define DIV          4
```

Αντίστοιχα με τον απαριθμητικό τύπο δηλώνω τον απαριθμητικό τύπο `choice` και τη μεταβλητή αυτού του τύπου `choice`[1]

```
enum choice {ADD=1, SUB, MUL, DIV}choice;
```

Και οι δύο παραπάνω δηλώσεις έχουν σαν αποτέλεσμα η πρόταση `switch` να πάρει τη μορφή

[1] Η C μας επιτρέπει να δηλώνουμε μεταβλητή με όνομα ίδιο με όνομα τύπου. Ο μεταγλωττιστής αναγνωρίζει από τα συμφραζόμενα αν αναφερόμαστε στον τύπο ή την μεταβλητή. Παρόλα αυτά καλό είναι να αποφεύγετε τέτοιες δηλώσεις.

```
switch(choice) {
   case ADD: πρόσθεσε τους αριθμούς;                          break;
   case SUB: αφαίρεσε τους αριθμούς;                          break;
   case MUL: πολλαπλασίασε τους αριθμούς                      break;
   case DIV: διαίρεσε τους αριθμούς                           break;
   default : ενημέρωσε το χρήστη ότι η επιλογή δεν υποστηρίζεται; break;
   }
```

7.6

Μια έκδοση του ζητούμενου προγράμματος έχει ως κατωτέρω

```
#include <stdio.h>

main()
{
int ch, num_of_spaces = 0;

printf("Δώσε μια πρόταση:\n");
ch = getchar();
while(ch != '\n') {
   if(ch == ' ')
        num_of_spaces++;
   ch = getchar();
   }
printf("Ο αριθμός των κενών είναι %d.\n", num_of_spaces);
exit(0);
}
```

Είναι σημαντικό να προσέξετε ότι η πρόταση *ch = getchar();* χρησιμοποιείται δύο φορές. Σίγουρα δεν είναι η καλύτερη επιλογή από πλευράς δόμησης. Αλλά μην ανησυχείτε, έχουμε και την πρόταση `do while` να δούμε, ίσως μας βοηθήσει στην καλύτερη δόμηση.

Αν το πρόγραμμά σας δεν μοιάζει με το παραπάνω, θα χρειαστεί να επανέλθετε στη θεωρία τη σχετική με την πρόταση `while`. Στη συνέχεια, προσπαθήστε να γράψετε από την αρχή τον κώδικα του προγράμματος.

Αν αναπτύξατε ίδιο ή παρόμοιο κώδικα, μπράβο σας.

7.7

Μια έκδοση του ζητούμενου προγράμματος έχει ως κατωτέρω

```
#include <stdio.h>

main()
{
int i,n;
double factorial;

printf("Δώσε ένα θετικό αριθμό:");
scanf("%d",&n);

factorial = 1;
for(i=2;i<=n; ++i)
   factorial *= i;
printf("\n Το παραγοντικό του %d είναι %.0f", n, factorial);
}
```

Συγκρίνοντας τον κώδικα με τον αντίστοιχο του παραδείγματος 5 είναι προφανές ότι ο βρόχος `for` πλεονεκτεί, γεγονός που ήταν αναμενόμενο, καθώς η φύση του προβλήματος επιβάλει απαριθμητή επαναλήψεων που συνοδεύεται από αρχικοποίηση και ανανέωση της τιμής του.

Αν δεν απαντήσατε σωστά, θα πρέπει να ανατρέξετε πάλι στην υποενότητα 7.2.5. και στο παράδειγμα 5. Στη συνέχεια, δοκιμάστε να δώσετε μια έκδοση με βρόχο `do`.

Αν απαντήσατε σωστά, μπράβο σας. Αφιερώστε όμως και εσείς λίγο χρόνο για την έκδοση με `do`.

7.8

Στη συνέχεια δίνεται μια έκδοση του ζητούμενου προγράμματος με χρήση της πρότασης `for`.

```
void func(int num[], int items)
{
for(i=0;i<items;i++)
   if (num[i]<0)
        num[i]++;
```

```
   else
        num[i] — = 2;
}
```

Προσέξτε τη στηλοθέτιση του πηγαίου κώδικα και τη μη χρησιμοποίηση αγκυλών για το σώμα της `for`, καθώς αυτό αποτελείται από μία πρόταση. Προσέξτε, επίσης, την έκφραση τερματισμού του βρόχου είναι `i<items` και όχι `i<=items`.

Αν ο κώδικας που δώσατε με πρόταση `for` δεν είναι παρόμοιος με τον παραπάνω, εντοπίστε τις διαφορές και προχωρήστε στη συγγραφή των εκδόσεων με `while` και `do`. Στη συνέχεια, γράψτε ένα πρόγραμμα που θα παίρνει από το χρήστη τα στοιχεία ενός πίνακα 10 ακεραίων και θα καλεί τη συνάρτηση που αναπτύξατε. Επιμείνετε στην κατανόηση των προτάσεων `for while` και `do`, καθώς αποτελούν τη βάση για τη διαμόρφωση της ροής των προγραμμάτων σας.

Αν οι τρεις εκδόσεις σας είναι σωστές, σας αξίζουν συγχαρητήρια, προχωρήστε όμως στη συγγραφή του προγράμματος που ανέφερα στην παραπάνω παράγραφο.

7.9

Στη συνέχεια, δίνεται μια έκδοση του ζητούμενου προγράμματος.

```
#include <stdio.h>
#define YEARS  3
#define MONTHS 12

float temp[YEARS][MONTHS]= {
                        {10, 12, 13,  14,  20,  30,  32,  36,  24,  20,  15,  10},
                        {12, 14, 15,  16,  22,  32,  34,  38,  26,  22,  17,  12},
                        { 8, 10, 11,  12,  18,  28,  30,  34,  22,  18,  13,  8},
                        };

float avg_temp[3];

void main()
{
int year, month;
float total, subtotal;
```

```
total = 0;
for(year=0;year<YEARS;year++){
   subtotal = 0;
   for(month=0;month<MONTHS;month++)
        subtotal+=temp[year][month];
   avg_temp[year] = subtotal/MONTHS;
   printf("Η μέση θερμοκρασία του %dου έτους είναι: %.2f\n",
           year,avg_temp[year]);
   total += avg_temp[year];
   }
printf("Η μέση θερμοκρασία των %d ετών είναι %.4f:\n",
        YEARS,total/YEARS);

for(month=0;month<MONTHS;month++){
   subtotal = 0;
   for(year=0;year<YEARS;year++)
        subtotal += temp[year][month];
   printf("Η μέση θερμοκρασία του %dου μήνα είναι: %.1f\n",
           subtotal/YEARS);
   }
}
```

7.10

Ανατρέξτε στη σελίδα 143 του [King '96] όπου θα βρείτε μια έκδοση του ζητούμενου προγράμματος. Ενδιαφέρον παρουσιάζει η χρήση της λέξης–κλειδί `typedef` που χρησιμοποιείται για να ορίσει ένα νέο όνομα για τον τύπο `int`. Η C δεν διαθέτει τύπο Boolean και για το λόγο αυτό χρησιμοποιούμε τον τύπο του ακεραίου. Η χρήση της πρότασης `typedef int Bool;` μας δίνει την ψευδαίσθηση ότι έχουμε πίνακα με στοιχεία τύπου Boolean. Στη συνέχεια, δίνεται μια έκδοση του ζητούμενου προγράμματος

```
#include <stdio.h>

#define TRUE  1
#define FALSE 0

typedef int Bool;
```

```
main()
{
Bool digit_seen[10] = {0};
int digit;
long int n;

printf("Δώσε ένα αριθμό: ");
scanf("%ld", &n);

while(n>0) {
   digit = n % 10;
   if(digit_seen[digit])
        break;
   digit_seen[digit] = TRUE;
   n /= 10;
   }
if (n>0)
   printf("Επανάληψη ψηφίου\n");
else
   printf("Μη επανάληψη ψηφίου\n");

return(0);
}
```

Αν δεν απαντήσατε σωστά, δεν πειράζει. Είναι μια άσκηση που απαιτεί καλό χειρισμό των προτάσεων ελέγχου ροής αλλά και του τελεστή `%`. Προσέξτε τη χρήση της πρότασης `break` για την έξοδο από το βρόχο όταν διαπιστωθεί μια επανάληψη.

Αν δεν απαντήσατε σωστά θα πρέπει να ανατρέξετε στην υποενότητα 7.2.3 για την πρόταση `while` και την 7.2.8 για την πρόταση `break` και, στη συνέχεια, επιχειρήστε να γράψετε εξ' αρχής τον κώδικα.

8.1

Στον πηγαίο κώδικα του σχήματος 8.1, παρατηρούμε τα ονόματα `a` και `b` να δηλώνονται αλλά και να χρησιμοποιούνται σε περισσότερα του ενός σημεία. Η εφαρμογή των κανόνων εμβέλειας είναι απαραίτητη για τον προσδιορισμό της αντιστοίχησης του ονόματος με το κατάλληλο αντικείμενο στη μνήμη και,

άρα, του αποτελέσματος του προγράμματος. Η πρώτη δήλωση στη γραμμή 3 δηλώνει τη μεταβλητή `a` σαν γενική μεταβλητή με εμβέλεια προγράμματος. Εμβέλεια προγράμματος έχει και η μεταβλητή `c` αλλά είναι ενεργή από το σημείο δήλωσής της (γραμμή 11) και κάτω, που σημαίνει πως δεν τη βλέπει η `main` και για το λόγο αυτό ο μεταγλωττιστής μας αναφέρει λάθος στην αναφορά αυτή. Αντίθετα, η `b` στη γραμμή 4 έχει εμβέλεια αρχείου, όπως προσδιορίζει η λέξη `static`. Τα `a` και `b` της `max` (γραμμή 12) έχουν εμβέλεια μπλοκ και αποκρύπτουν για το σώμα της `max` τις γενικές μεταβλητές `a` και `b`. Η γραμμή 6 αποδίδει στα `a` και `b` τις τιμές 12 και 11, αντίστοιχα. Η γραμμή 8 καλεί τη συνάρτηση `max` και δίνει στα τυπικά ορίσματα `a` και `b` της `max` τις τιμές 17 και 11, αντίστοιχα. Η `max` επιστρέφει το 17, το οποίο και τυπώνεται από την `printf`. Στη συνέχεια, καλείται η συνάρτηση `func` και ανατίθεται η τιμή 23 στο τυπικό της όρισμα `x` το οποίο έχει εμβέλεια μπλοκ. Η τοπική μεταβλητή `b` που δηλώνεται στη γραμμή 16, αποκρύπτει από το σώμα της `func` τη γενική μεταβλητή `b`. Δεν συμβαίνει όμως το ίδιο και για τη γενική μεταβλητή `a`, η οποία είναι ορατή από το σώμα της `func`.

Αφού αφαιρέσουμε την αναφορά στη μεταβλητή `c` από την `printf` της `main` (γραμμή 7), η έξοδος του προγράμματος θα είναι η παρακάτω:

```
a:11    b:12        max(b+5,a):17
a:11    b:20        c:13        x:23        max(x,b):23
```

Αν τα αποτελέσματά σας δεν είναι σωστά, επιμείνατε πολύ στο θέμα της εμβέλειας ξαναδιαβάζοντας την αντίστοιχη θεωρία. Είναι ένα από τα σημαντικότερα θέματα που σχετίζονται με τη σωστή οργάνωση του πηγαίου κώδικα.

8.5

Μια έκδοση της ζητούμενης συνάρτησης έχει την παρακάτω μορφή

```
float power(float x, int n)
{
if (n==0)
   return(1.0);
else if (n>0)
   return(x * power(x,n–1));
else
   return (1.0 / power(x,–n));
```

```
}
```

Προσέξτε τη συνθήκη τερματισμού της αναδρομής, η οποία ορίζεται για n ίσο με 0 και για την οποία η συνάρτηση επιστρέφει το 1.0. Το παράδειγμα 8.11 του [Rojiani '96] δίνει μια έκδοση του προγράμματος με ταυτόχρονο σχολιασμό της διαδικασίας ανάπτυξης.

8.6

Ο ορισμός της συνάρτησης fibonacci με χρήση αναδρομικότητας έχει ως ακολούθως

```
float fibonacci(int n)
{
if(n<=2)
   return(1.0);
else
   return(fibonacci(n–1) + fibonacci(n–2));
}
```

Στο παράδειγμα 8.12 του [Rojiani '96] μπορείτε να βρείτε μια έκδοση του ζητούμενου προγράμματος και σχολιασμό του παραπάνω ορισμού της συνάρτησης fibonacci.

9.1

Η έκδοση που σας δίνουμε στη συνέχεια, χρησιμοποιεί τις τρεις διαφορετικές εκδόσεις της `read_line` που δώσαμε στις ασκήσεις αυτοαξιολόγησης 3 και 4, καθώς και τη συνάρτηση `report_circle` για την εμφάνιση των στοιχείων του κύκλου.

```
#include <stdio.h>
#define    PI  3.14

struct point {
   float x;
   float y;
   };

struct circle{
   struct point p;
```

```
   float r;
   };

struct circle read_circle(struct circle c1);
struct circle read_circle2(void);
void read_circle3(struct circle *c1);
void report_circle(struct circle c);

main()
{
struct circle c1;

c1 = read_circle(c1);
report_circle(c1);
c1 = read_circle2();
report_circle(c1);
read_circle3(&c1);
report_circle(c1);
}

struct circle read_circle(struct circle c1)
{
printf("Dwse sintetagmenes kentrou: ");
scanf("%f %f", &c1.p.x, &c1.p.y);
printf("Dwse aktina: ");
scanf("%f", &c1.r);
return c1;
}

struct circle read_circle2(void)
{
struct circle c1;
printf("Dwse sintetagmenes kentrou: ");
scanf("%f %f", &c1.p.x, &c1.p.y);
printf("Dwse aktina: ");
scanf("%f", &c1.r);
return c1;
```

```
}

void read_circle3(struct circle *c1)
{
printf("Dwse sintetagmenes kentrou: ");
scanf("%f %f", &c1->p.x, &c1->p.y);
printf("Dwse aktina: ");
scanf("%f", &c1->r);
}

void report_circle(struct circle c)
{
printf("x: %f\ty: %f\tr: %f\n", c.p.x, c.p.y, c.r);
printf("circumference: %f\tarea: %f\n", 2*PI*c.r, PI*c.r*c.r);
printf("\n");
}
```

Προσέξτε τη δήλωση της σταθεράς `PI`, καθώς και τη συμβολή της συνάρτησης `report_circle` στην αποφυγή του πλεονασμού και τη βελτίωση της αναγνωσιμότητας της `main`.

10.1

Μια έκδοση του ζητούμενου προγράμματος έχει ως κατωτέρω:

```
#include <stdio.h>

void ComputeAve(int num1, int num2, int *sum, int *average);

main()
{
int num1, num2;
int sum, average;

num1 =100;
num2 = 200;
ComputeAve(num1, num2, &sum, &average);
printf("sum is %d and average is %d\n", sum, average);
}
```

```
void ComputeAve(int num1, int num2, int *sum, int *average)
{
   *sum = num1 + num2;
   *average = *sum / 2;
}
```

Η συνάρτηση πρέπει να επιστρέφει δύο τιμές, γεγονός που αποκλείει τη χρήση του μηχανισμού επιστρεφόμενης τιμής της συνάρτησης της C. Για την επιστροφή του αθροίσματος και του μέσου όρου ακολουθείται τακτική ανάλογη της Pascal, αλλά με χρήση όχι της var αλλά των δεικτών.

Αν δε δώσατε μια λειτουργούσα έκδοση σε γενικές γραμμές σύμφωνη με την παραπάνω, είναι ένδειξη πως δεν έχετε εξοικειωθεί αρκετά με τις έννοιες των κεφαλαίων 6 και 8, στα οποία θα πρέπει να ανατρέξετε.

Αν τα καταφέρατε, σας αξίζουν συγχαρητήρια.

Βιβλιογραφία ελληνική

[Horowitz ’84]

«Βασικές αρχές γλωσσών προγραμματισμού», Εκδόσεις Κλειδάριθμος, 1993.

Το βιβλίο είναι μετάφραση της δεύτερης Αμερικάνικης έκδοσης του «Fundamentals of Programming Languages» Ellis Horowitz, Computer Science Press, 1984.

Αποτελεί μια πολύ καλή και, κατά τα γνωστά μου, μοναδική στην Ελληνική γλώσσα πηγή, αναφορικά με τις βασικές αρχές των γλωσσών προγραμματισμού. Το βιβλίο είναι κατάλληλο για ανάγνωση μόνο επιλεκτικών τμημάτων από σπουδαστές που μαθαίνουν την πρώτη τους γλώσσα προγραμματισμού. Είναι όμως απαραίτητο για σπουδαστές που γνωρίζουν και έχουν χρησιμοποιήσει εκτεταμένα μια τουλάχιστον γλώσσα προγραμματισμού.

[Kernighan ’88]

Η δεύτερη έκδοση του βιβλίου *«The C Programming Language»* κυκλοφόρησε το 1988 καλύπτοντας πλέον την ANSI C. Ελληνική έκδοση σε μετάφραση από τον Κλειδάριθμο 1990.

Αποτελεί το πλέον έγκυρο βιβλίο μετά το ISO/IEC 9899, πολύ καλό για αναφορά αλλά πολύ δύσκολο για να χρησιμοποιηθεί για εκμάθηση της γλώσσας

[Waite ’90]

Mitchell Waite, Stephen Prata *«New C Primer Plus»* Waite Group Inc 1990.

Η δεύτερη έκδοση σε Ελληνική μετάφραση από τον Γκιούρδα 1991 με τίτλο «C:Βήμα–προς–Βήμα».

Εγχειρίδιο της C.

Ελληνική μετάφραση από τον Γκιούρδα (1998) του *«Teach Yourself in 21 Days»* 4th edition, 1997 Sams Publishing.

Βιβλιογραφία Ξενόγλωσση

[Aho–Ullman’72]

Aho, A., Ullman, J. «*The theory of Parsing, Translation and Compiling*» Prentice–Hall 1972.

[**ANSI ’88]**

Περισσότερες πληροφορίες για το πρωτότυπο ISO/IEC 9899 μπορείτε να βρείτε στο δια–δίκτυο στη διεύθυνση
http://wwold.dkuug.dk/JTC1/SC22/WG14/www/standards

[Bergin ’96]

Bergin Thomas, Gibson Richard, «*History of Programming Languages*», Addison Wesley 1996.

Το βιβλίο αποτελεί σημαντική προσπάθεια συγκέντρωσης πολλών σημαντικών γλωσσών και συνιστάται για κάθε αναγνώστη που εμπλέκεται στη χρήση ή ανάπτυξη γλωσσών προγραμματισμού. Είναι βασισμένο στα πρακτικά του δευτέρου ACM SIGPLAN συνεδρίου «History of Programming Languages». Περιλαμβάνει περίληψη των πρακτικών του συνεδρίου, καθώς και άρθρα των σημαντικότερων συντελεστών στον χώρο των γλωσσών προγραμματισμού: Frederick Brooks, Alain Colmerauer, Richard Gabriel, Ralph Griswold, Per Brinch, Hansen, Alan Kay, C. H. Lindsey, Barbara Liskov, Richard Nance, Elizabeth Rather, Dennis Ritchie, Jean Sammet, Guy Steele, Bjarne Stroustrup, William Whitaker, και Niklaus Wirth. Μεταξύ των γλωσσών που ιδιαίτερα αναφέρονται είναι οι C, C++, Smalltalk,Pascal, Ada, Prolog, Lisp, ALGOL 68, FORMAC, CLU, Icon, Forth, και Concurrent Pascal

[Boehm ’76]

Boehm W. Barry, «*Software Engineering*» IEEE Trans. Comput., vol. C–25, pp 1226–1241, Dec. 1976.

[Boehm ’88]

Boehm W. Barry, «*A Spiral Model of Software Development and Enhancement*» IEEE Computer, vol. 21, no 5, p.61–72, May 1988.

Corriveau Jean Paul, «*A step–by–step Guide to C Programming*», Prentice Hall.

[Darnell 1991]

Darnell P., Margolis P.: «*C: A Software Engineering Approach*», Springer–Verlag, New York 1991.

[Hanly ’99]

Hanly J.R. and Koffman E.B., «*Problem Solving & Problem Design in C*», third edition, Addison–Wesley, 1999.

Το βιβλίο ακολουθεί πολύ καλή προσέγγιση, καθώς παρουσιάζοντας βασικές αρχές προγραμματισμού και επίλυσης προβλημάτων διαμέσου της C, δίνει έμφαση στα υψηλού επιπέδου χαρακτηριστικά της C. Μπορεί να χρησιμοποιηθεί σαν κύριο βοήθημα σε Πανεπιστημιακού επιπέδου εισαγωγικά μαθήματα προγραμματισμού.

[Horowitz ’84]

Horowitz Ellis «*Fundamentals of Programming Languages*», Computer Science Press, 1984.

Αποτελεί μια πολύ καλή πηγή αναφορικά με τις βασικές αρχές των γλωσσών προγραμματισμού. Το βιβλίο είναι κατάλληλο για ανάγνωση μόνο επιλεκτικών τμημάτων από σπουδαστές που μαθαίνουν την πρώτη τους γλώσσα προγραμματισμού. Είναι όμως απαραίτητο για σπουδαστές που γνωρίζουν και έχουν χρησιμοποιήσει εκτεταμένα μια τουλάχιστον γλώσσα προγραμματισμού.

[Horowitz ’95]

Horowitz Ellis, «*Fundamentals of Programming Languages*», Third edition, Computer Science Press 1995.

[Hsia ’88]

Hsia Y.T., Ambler A., «*Programming Through Pictorial transformations*»,Proc. Int’l Conf. Computer Languages 88, IEEE CS Press, Los Alamitos, Calif., Order No. 1988, pp.10–16.

[Kernighan ’81]

Kernighan W. Brian, «*Why Pascal is not my favorite programming language*» AT&T Bell Laboratories, Computer Science Technical Report No.100.

Μπορείτε να βρείτε ένα αντίγραφο του άρθρου σε ηλεκτρονική μορφή στη διεύθυνση http://netlib.bell–labs.com/cm/cs/cstr/100.ps.gz

Στο άρθρο υπάρχουν αρκετές αναφορές σε εργασίες αξιολόγησης και σύγκρισης των δύο γλωσσών.

[Kernighan ’88]

Η δεύτερη έκδοση του βιβλίου «*The C Programming Language*» κυκλοφόρησε το 1988 καλύπτοντας πλέον την ANSI C.

Αποτελεί το πλέον έγκυρο βιβλίο μετά το ISO/IEC 9899, πολύ καλό για αναφορά αλλά πολύ δύσκολο για να χρησιμοποιηθεί για εκμάθηση της γλώσσας

[King ’96]

King K.N., «*C Programming: A modern approach*», W.W.Norton & Company, Inc.

[Koffman ’95]

Koffman B. Elliot «*Pascal*» 5th Edition, Addison–Wesley Publishing Company.

Το κλασικό βιβλίο της Pascal, απόλυτα σύμφωνο με το βασικό κορμό σειράς μαθημάτων της επιστήμης των υπολογιστών της ACM. Καλύπτει με επιτυχία, όχι μόνο τη γλώσσα Pascal, αλλά παρεμφερή θέματα, όπως problem solving και software engineering.

[Maulsby ’89]

Maulsby D.L., Witten H., «*Inducing Programs in a Direct–Manipulation Environmant*», Proc. CHI’89, ACM Press, New York, 1989, pp. 57–62.

[Ritchie ’93]

Ritchie M. Dennis «*The Development of the C Language*», ACM SIGPLAN Notices, March 93, p.207.

Το άρθρο δίνει μια πολύ καλή αναφορά στην εξέλιξη της γλώσσας C.

[Rojiani '96]

Rojiani K.B., «*Programming in C with numerical methods for Engineers*», Prentice–Hall 1996.

[Sethi '97]

Sethi Ravi, «*Programming Languages: Concepts and Constructs*» 2nd Edition, Addison Wesley 1996. Reprinted with corrections, April 1997

[Unicode]

Το unicode που τυποποιήθηκε σαν ISO 10646 πρότυπο βασίζει την κωδικοποίηση των χαρακτήρων σε ένα σχήμα 16 bit, αυξάνοντας με τον τρόπο αυτό κατά πολύ τον αριθμό των χαρακτήρων που μπορεί να αναπαραστήσει. Πληροφορίες για το unicode μπορείτε να βρείτε στο διαδίκτυο στη διεύθυνση http://www.unicode.org/.

[Waite '90]

Waite Mitchell, Prata Stephen «*New C Primer Plus*» Waite Group Inc 1990.

[Wexelblat '81]

Wexelblat L. Richard «*History of Programming Languages*», Los Angeles, 1981.

Περιέχει τα πρακτικά του πρώτου ACM SIGPLAN συνεδρίου «History of Programming Languages» Los Angeles on June 1–3, 1978. Καταγράφει τις γλώσσες προγραμματισμού που δημιουργήθηκαν το τέλος της δεκαετίας '60 (1967) και παρέμεναν σε χρήση μέχρι το 1977, επηρεάζοντας σημαντικά την εξέλιξη του προγραμματισμού. Περιέχονται άρθρα ερευνητών που διαδραμάτισαν σημαντικό ρόλο στην ανάπτυξη και χρήση των γλωσσών : ALGOL, APL, APT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/I, SIMULA, AND SNOBOL.

[Wirth '76]

Wirth N., «*Algorithms + Data Structures = Programs*», Prentice–Hall, 1976.

Γλωσσάρι όρων

– –	Μοναδιαίος τελεστής της C. Μειώνει κατά 1 την τιμή της μεταβλητής στην οποία εφαρμόζεται.
!	Ο λογικός τελεστής NOT της C.
&	Μοναδιαίος τελεστής της C. Δίνει τη διεύθυνση της μεταβλητής στην οποία εφαρμόζεται.
&&	Ο λογικός τελεστής AND της C.
*	Σαν μοναδιαίος τελεστής της C (τελεστής περιεχομένου) εφαρμόζεται μόνο σε δείκτη και έχει σαν αποτέλεσμα το περιεχόμενο της θέσης μνήμης που δείχνει ο δείκτης.
? :	Ο υποθετικός τελεστής της C.
||	Ο λογικός τελεστής OR της C.
++	Μοναδιαίος τελεστής της C. Αυξάνει κατά 1 την τιμή της μεταβλητής στην οποία εφαρμόζεται.
=	Τελεστής ανάθεσης της C. Εφαρμόζεται σε δύο τελεστέους σύμφωνα με την ενθεματική σημειολογία και έχει σαν αποτέλεσμα την ανάθεση της τιμής του δεξιού τελεστέου στον αριστερό.
= =	Τελεστής της C για έλεγχο ισότητας.
abstract syntax tree	Δένδρο αφηρημένης σύνταξης.
abstraction	Αφαιρετικότητα.
ALGOL	Συντομογραφία του algorithmic language. Υψηλού επιπέδου γλώσσα προγραμματισμού. Εμφανίστηκε σαν διάδοχος της FORTRAN και κυριάρχησε στη δεκαετία του ’60.
ANSI	Συντομογραφία του American National Standards Institute. Αμερικάνικος εθνικός οργανισμός που καθορίζει και δημοσιεύει τα αμερικάνικα βιομηχανικά πρότυπα.
ANSI C	Ο ακριβής ορισμός της C όπως ορίστηκε το 1983 από τον ANSI.

applicative order	Εφαρμοστική σειρά.
ASCII κώδικας	Συντομογραφία του American Standard Code for Information Interchange. Κώδικας που χρησιμοποιεί 7 ή 8 δυαδικά ψηφία για την αναπαράσταση αλφαβητικών, αριθμητικών και χαρακτήρων ελέγχου, καθώς και ειδικών συμβόλων.
Basic	Συντομογραφία του Beginners All–purpose Symbolic Instruction Code. Υψηλού επιπέδου γλώσσα προγραμματισμού, η οποία παρότι σχεδιάστηκε για εκπαίδευση αρχαρίων στο προγραμματισμό, εξελίχθηκε σε γενικού σκοπού και αρκετά δημοφιλή γλώσσα.
binary operator	Δυαδικός τελεστής.
BNF	Συντομογραφία του Backus–Naur–Form. Σημειογραφία που αναπτύχθηκε το 1963 για να περιγράψει τη σύνταξη της γλώσσας ALGOL60. Αποτελεί με τις επεκτάσεις της (π.χ., EBNF) μια από τις πλέον δημοφιλείς σημειογραφίες μετα–σύνταξης για τον προσδιορισμό του συντακτικού των γλωσσών προγραμματισμού.
break	Λέξη – κλειδί της C που διευκολύνει τον έλεγχο της ροής στην πρόταση switch καθώς και τη διαχείριση ειδικών περιπτώσεων μέσα σε βρόχους επανάληψης.
call by reference	Κλήση κατ' αναφορά.
call by value	Κλήση κατ' αξία, κλήση κατά τιμή.
char	Λέξη–κλειδί της C που αναπαριστά τον τύπο του χαρακτήρα.
compiler	Μεταγλωττιστής.
conditional loop	Επανάληψη υπό συνθήκη.
continue	Λέξη–κλειδί της C που διευκολύνει τη διαχείριση ειδικών περιπτώσεων μέσα σε βρόχους επανάληψης.

counting loop Απαριθμούμενη επανάληψη.

data abstraction Αφαιρετικότητα στα δεδομένα.

data type Τύπος δεδομένων.

declaration part Το τμήμα εκείνο ενός προγράμματος Pascal που περιλαμβάνει τις δηλώσεις των σταθερών και των μεταβλητών αλλά και τους ορισμούς των συναρτήσεων (functions) και διαδικασιών (procedures).

define Εντολή του προεπεξεργαστή της C.

do while Λέξη–κλειδί της C (και άλλων γλωσσών) για την επανειλημμένη εκτέλεση μιας ή συνόλου προτάσεων όσο μια συνθήκη είναι αληθής (conditional loop).

double Λέξη–κλειδί της C που αναπαριστά τον τύπο των αριθμών κινητής υποδιαστολής διπλής ακρίβειας.

duration (of a variable) Διάρκεια μεταβλητής.

EBCDIC Συντομογραφία του Extended Binary Coded Decimal Interchange Code. Κώδικας για αναπαράσταση χαρακτήρων που χρησιμοποιείται κύρια από την IBM.

float Λέξη–κλειδί της C που αναπαριστά τον τύπο των αριθμών κινητής υποδιαστολής απλής ακρίβειας.

for Λέξη–κλειδί της C (και άλλων γλωσσών) για τη δημιουργία της πρότασης προκαθορισμένου αριθμού επαναλήψεων (counting loop).

FORTRAN Συντομογραφία του FORmula TRANslation. Η πρώτη χρονολογικά υψηλού επιπέδου γλώσσα προγραμματισμού. Σχεδιάστηκε για να διευκολύνει κύρια την έκφραση μαθηματικών τύπων. Η γλώσσα εξακολουθεί και σήμερα να αποτελεί την πιο δημοφιλή επιλογή μεταξύ των επιστημόνων και των μηχανικών.

function prototype Πρωτότυπο συνάρτησης.

goto	Λέξη–κλειδί της C για ρητή διακλάδωση σε συγκεκριμένη θέση. Η χρήση της είναι αντίθετη με τις αρχές του δομημένου προγραμματισμού.
identifier	Αναγνωριστής.
if else	Λέξεις–κλειδί της C (και άλλων γλωσσών) για τη δημιουργία της πρότασης διακλάδωσης υπό συνθήκη.
include	Εντολή του προεπεξεργαστή. Έχει σαν αποτέλεσμα την συμπερίληψη του αρχείου του οποίου το όνομα ακολουθεί.
int	Λέξη–κλειδί της C που αναπαριστά τον τύπο του ακεραίου.
ISO	Συντομογραφία του International Standards Organization. Διεθνής οργανισμός τυποποίησης.
K&R C	Συντομογραφία του Kernighan and Ritchie C. Ο ορισμός της γλώσσας C όπως είχε οριστεί από το σύγγραμμα «The C Programming Language» των B. Kernighan και D. Ritchie.
keyword	Λέξη–κλειδί.
linker	Συνδέτης.
local variable	τΤπική μεταβλητή.
machine language	Γλώσσα μηχανής.
Modula	Γλώσσα προγραμματισμού υψηλού επιπέδου που αναπτύχθηκε από τον Niklaus Wirth στα τέλη της δεκαετίας 70 με στόχο να διορθώσει κάποιες από τις αδυναμίες της Pascal. Είναι κατάλληλη για εκμάθηση προγραμματισμού, για ανάπτυξη μεγάλων εφαρμογών, σύμφωνα με τις αρχές του Software Engineering και για ενσωματωμένα συστήματα πραγματικού χρόνου (real time embedded systems).
Object Pascal	Επέκταση της Pascal με κατασκευές κατάλληλες για την υποστήριξη του αντικειμενοστρεφούς προγραμματισμού.

OO Object–Oriented. Αντικειμενοστρεφής.

OO programming Object–Oriented programming. Αντικειμενοστρεφής προγραμματισμός.

Pascal Υψηλού επιπέδου γλώσσα προγραμματισμού που, αν και σχεδιάστηκε από τον Nicklaus Wirth στις αρχές της δεκαετίας του ’70 σαν εκπαιδευτική γλώσσα, εξελίχθηκε σε γενικού σκοπού προστακτική γλώσσα. Ιδιαίτερα δημοφιλής, επηρέασε σε μεγάλο βαθμό επόμενες γλώσσες προστακτικού προγραμματισμού, μεταξύ των οποίων οι Modula, Concurrent Pascal και Ada.

printf Συνάρτηση της βασικής βιβλιοθήκης της C για έξοδο μορφοποιημένης πληροφορίας στην κύρια έξοδο.

procedure Διαδικασία. Είναι, όπως και η συνάρτηση, μια ανεξάρτητη μονάδα (program unit or module) που περιλαμβάνει ένα σύνολο από προτάσεις που επιλύουν ένα συγκεκριμένο υπο–πρόβλημα. Η βασική της διαφορά από τη συνάρτηση εντοπίζεται στο γεγονός ότι δεν είναι απαραίτητο να επιστρέφει ρητά κάποια τιμή, μπορεί όμως και να επιστρέψει περισσότερες από μία τιμές και να εκδηλώνει παρενέργειες. Αντίθετα, οι συναρτήσεις λαμβάνουν συνήθως περισσότερες της μίας τιμές και επιστρέφουν πίσω μία τιμή, χωρίς την ύπαρξη παρενεργειών (πλευρικών φαινομένων).

program body Το τμήμα εκείνο ενός προγράμματος Pascal που αρχίζει με τη δεσμευμένη λέξη begin, περιλαμβάνει ένα σύνολο από προτάσεις που προσδιορίζουν το έργο του προγράμματος και τελειώνει με τη δεσμευμένη λέξη end.

programming paradigm/style Μορφή ή στυλ προγραμματισμού. Είναι μια συλλογή από έννοιες, οι οποίες, προσδιορίζοντας ένα ορισμένο τρόπο σκέψης και, άρα, έκφρασης της

	λύσης, επηρεάζουν το σχεδιασμό των προγραμμάτων.
recursive function	Αναδρομική συνάρτηση.
reserved word	Δεσμευμένη λέξη.
return	Η λέξη–κλειδί της C που χρησιμοποιείται για να μεταφέρει τον έλεγχο από μια συνάρτηση πίσω στο σημείο από το οποίο αυτή κλήθηκε.
scanf	Συνάρτηση της βασικής βιβλιοθήκης της C για είσοδο μορφοποιημένης πληροφορίας στην κύρια είσοδο.
scope rules	Κανόνες εμβέλειας.
semantic error	Σημασιολογικό σφάλμα.
short circuit evaluation	Υπολογισμός περιορισμένης έκτασης.
side effect	Παρενέργεια.
sizeof	Μοναδιαίος τελεστής της C. Δίνει τον αριθμό των bytes που η τιμή μιας έκφρασης ή ενός τύπου δεδομένων καταλαμβάνει στη μνήμη.
Smalltalk	Αντικειμενοστρεφής γλώσσα προγραμματισμού. Σχεδιάστηκε στο ερευνητικό κέντρο Palo Alto της Xerox στις αρχές της δεκαετίας του ’70. Οι βασικότερες έννοιές της αποδίδονται στον Alan Kay που χρησιμοποίησε πολλά στοιχεία από τις Simula, LISP και SketchPad. Παρά τις αρχικές προβλέψεις δεν κατάφερε να γίνει αρκετά δημοφιλής κύρια, λόγω της εμφάνισης της C++ και της Java.
standard C library	Βασική βιβλιοθήκη της C.
statement	Πρόταση.
static	Λέξη–κλειδί της C για τον προσδιορισμό της εμβέλειας συναρτήσεων και καθολικών μεταβλητών, αλλά και διάρκειας ζωής τοπικών μεταβλητών.
string	Αλφαριθμητικό.
string.h	Αρχείο επικεφαλίδας της βασικής βιβλιοθήκης της C.

strlen Συνάρτηση της βασικής βιβλιοθήκης της C. Δίνει τον αριθμό των χαρακτήρων του αλφαριθμητικού που δέχεται σαν όρισμα.

struct Λέξη–κλειδί της C που επιτρέπει τον ορισμό νέων τύπων για τη διαχείριση ομάδων δεδομένων διαφορετικού τύπου που έχουν σχέση μεταξύ τους.

structured programming Δομημένος προγραμματισμός.

switch case Λέξεις–κλειδί της C για τη δημιουργία της πρότασης υπό συνθήκη διακλάδωσης που υποστηρίζει την επιλογή μιας από ένα σύνολο αμοιβαία αποκλειόμενων επιλογών.

syntax error Συντακτικό σφάλμα.

ternary operator Τριαδικός τελεστής.

unary operator Μοναδιαίος τελεστής.

Unicode Αποτελεί το διεθνές πρότυπο κωδικοποίησης χαρακτήρων. Είναι πλήρως συμβατό με το διεθνές πρότυπο ISO/IEC 10646 και βασίζει την κωδικοποίηση των χαρακτήρων σε ένα σχήμα 16 δυαδικών ψηφίων. Περιέχει πρόσθετη πληροφορία για τους χαρακτήρες αλλά και τη χρήση τους.

variable Μεταβλητή.

while Λέξη–κλειδί της C (και άλλων γλωσσών) για την επανειλημμένη εκτέλεση μιας ή συνόλου προτάσεων, όσο μια συνθήκη είναι αληθής (conditional loop).

αλφαριθμητικό (string) δεδομένο που αποτελείται από μια ακολουθία χαρακτήρων.

αναγνωριστής (identifier) λεκτική μονάδα που κατασκευάζει ο προγραμματιστής και τη χρησιμοποιεί, συνήθως, σαν όνομα μιας δικής του κατασκευής με στόχο να την αναγνωρίζει μοναδιαία από το σύνολο των κατασκευών του προγράμματος.

αναδρομική συνάρτηση Συνάρτηση που καλεί τον εαυτό της.

Αντικειμενοστρεφής προγραμματισμός Μορφή/ στυλ προγραμματισμού όπου το κυρίαρχο δομικό στοιχείο είναι το αντικείμενο (object), το δε σύστημα δομείται σαν ένα σύνολο από αντικείμενα τα οποία αλληλεπιδρούν μεταξύ τους με ανταλλαγή μηνυμάτων.

ΑΠ Αντικειμενοστρεφής Προγραμματισμός. Αντικειμενοστρεφής Προσέγγιση.

αρθρωτός σχεδιασμός Τεχνική σχεδιασμού προγραμμάτων που βασίζει την ανάπτυξη σύνθετων συστημάτων λογισμικού στην τμηματοποίηση των σύνθετων διεργασιών σε επιμέρους απλούστερες διεργασίες ή αντίστοιχα την τμηματοποίηση του προγράμματος σε επιμέρους τμήματα (modules).

αφαιρετικότητα (abstraction). Αποτελεί μια απλοποιημένη περιγραφή ή τεκμηρίωση που δίνει έμφαση σε ορισμένα χαρακτηριστικά ενώ ταυτόχρονα αποσιωπεί άλλα.

βαθμωτός τύπος Τύπος του οποίου οι τιμές βρίσκονται κατά μήκος μιας γραμμικής κλίμακας. Π.χ., τύπος ακέραιου, πραγματικού, χαρακτήρα.

βασική βιβλιοθήκη της C Η βιβλιοθήκη συναρτήσεων που περιέχει το σύνολο των συναρτήσεων που το ANSI C πρότυπο όρισε σαν βασικές και πρέπει να συνοδεύουν κάθε μεταγλωττιστή της C που είναι ANSI συμβατός.

βιβλιοθήκη Μια συλλογή από συχνά χρησιμοποιούμενες μονάδες κώδικα που, συνήθως, έχουν τη μορφή συναρτήσεων, ρουτινών ή κλάσεων.

γενική μεταβλητή Καθολική μεταβλητή.

γλώσσα μηχανής Το σύνολο των εντολών που κατανοεί και μπορεί να εκτελέσει ο επεξεργαστής του υπολογιστή.

δένδρο αφηρημένης σύνταξης Δένδρο που δείχνει τη συντακτική δομή μιας έκφρασης ανεξάρτητα από τη σημειολογία σύνταξής της και χρησιμοποιείται από το μεταγλωττιστή για τον υπολογισμό της τιμής της.

δεσμευμένη λέξη Λεκτική μονάδα της οποίας η σημασία καθορίζεται από τους κανόνες της γλώσσας και δεν μπορεί να αλλάξει από το χρήστη.

δήλωση μεταβλητής Πρόταση που γνωστοποιεί στον μεταγλωττιστή το συμβολικό όνομα και το σύνολο των ιδιοτήτων της μεταβλητής.

διάρκεια μεταβλητής Το χαρακτηριστικό εκείνο που προσδιορίζει τον χρόνο που το όνομα της μεταβλητής είναι συνδεδεμένο με τη θέση μνήμης που περιέχει την τιμή της μεταβλητής.

δομημένος προγραμματισμός Ο προγραμματισμός με κατάλληλες κατασκευές ελέγχου ροής, οι οποίες διασφαλίζουν τη δόμηση του πηγαίου κώδικα, ώστε να είναι εύκολο να κατανοήσουμε τι κάνει το πρόγραμμα.

επιστρεφόμενη τιμή Η πληροφορία που επιστρέφει μια συνάρτηση, όταν επιστρέφει τον έλεγχο στο σημείο από το οποίο κλήθηκε.

εφαρμοστική σειρά Μέθοδος υπολογισμού τιμής εκφράσεων κατά την οποία για τον υπολογισμό εκφράσεων της μορφής *τελ1* ***τελεστής*** *τελ2* υπολογίζονται και οι δύο τελεστέοι τελ1 και τελ2 και, στη συνέχεια, εφαρμόζεται ο τελεστής στα αποτελέσματα.

καθολική μεταβλητή Μεταβλητή η οποία είναι προσπελάσιμη από όλα τα σημεία του προγράμματος.

κανόνες εμβέλειας Το σύνολο των κανόνων μιας γλώσσας που προσδιορίζουν την κατασκευή στην οποία αντιστοιχεί κάθε εμφάνιση ενός αναγνωριστή (identifier) στον πηγαίο κώδικα. Πιο απλά, κανόνες που προσδιορίζουν το τμήμα του πηγαίου κώδικα στο οποίο ένα όνομα είναι προσβάσιμο.

Λέξη–κλειδί Λεκτική μονάδα που από μόνη της ή με άλλες λεκτικές μονάδες σχηματίζει κάποια γλωσσική κατασκευή.

μεταβλητή Κατασκευή του προγραμματιστή που χαρακτηρίζεται από ένα συμβολικό όνομα, ένα σύνολο ιδιοτήτων, μια αναφορά και μία τιμή. Συνήθως, μόνο η τιμή της μεταβλητής μπορεί να αλλάζει στο χρόνο εκτέλεσης.

μεταγλωττιστής Ειδικό πρόγραμμα υπολογιστή για τη μετάφραση σε γλώσσα μηχανής, προγραμμάτων γραμμένων σε υψηλού επιπέδου γλώσσα προγραμματισμού όπως C, Pascal, Ada, FORTRAN.

μοναδιαίος τελεστής Τελεστής που εφαρμόζεται πάνω σε ένα μόνο τελεστέο π.χ., ο τελεστής προσήμου + της C.

παραγόμενος τύπος Τύπος που δημιουργείται από τον προγραμματιστή με την κατάλληλη αξιοποίηση του αντίστοιχου μηχανισμού ορισμού νέων τύπων που η γλώσσα διαθέτει. Προσφέρει πιο αποτελεσματική διαχείριση των δεδομένων της εφαρμογής αυξάνοντας, όχι μόνο την αναγνωσιμότητα του προγράμματος, αλλά και την αξιοπιστία του.

παρενέργεια (side effect) η αλλαγή της τιμής μιας μεταβλητής που προκαλείται χωρίς τη χρήση τελεστή ανάθεσης.

πραγματικό όρισμα (actual argument) η πληροφορία (τιμή ή διεύθυνση μεταβλητής) που μεταφέρεται στην καλούμενη συνάρτηση με την κλήση της.

προεπεξεργαστής (preprocessor) ειδικό πρόγραμμα που επεξεργάζεται τον πηγαίο C κώδικα πριν από το μεταγλωττιστή. Αναγνωρίζει ορισμένες εντολές και προκαλεί ένα σύνολο από τροποποιήσεις στον πηγαίο κώδικα πριν αρχίσει το έργο της μεταγλώττισης.

πρόταση Μία πλήρης εντολή (command) προς τον υπολογιστή που προσδιορίζει την εκτέλεση συγκεκριμένου έργου.

πρωτότυπο συνάρτησης Μια πρόταση που προσδιορίζει τη διεπαφή της συνάρτησης, με άλλα λόγια, τον τρόπο αναφοράς

στη συνάρτηση. Η ύπαρξή της είναι απαραίτητη πριν από τη χρήση της συνάρτησης.

σημασιολογικό σφάλμα Σφάλμα που οφείλεται στην κακή απόδοση της λύσης του προβλήματος. Τα σημασιολογικά σφάλματα αναγνωρίζονται συνήθως στο χρόνο εκτέλεσης, καθώς δεν αναγνωρίζονται από το μεταγλωττιστή.

σημειολογία ένθεσης Η σημειολογία κατά την οποία ο τελεστής τοποθετείται μεταξύ των δεδομένων στα οποία ενεργεί.

σημειολογία επίθεσης Η σημειολογία κατά την οποία ο τελεστής τοποθετείται μετά από τα δεδομένα στα οποία ενεργεί.

σημειολογία πρόθεσης Η σημειολογία κατά την οποία ο τελεστής τοποθετείται πριν από τα δεδομένα στα οποία ενεργεί.

συναθροιστικός τύπος Τύπος που δημιουργείται συνδυάζοντας έναν ή περισσότερους βαθμωτούς τύπους. Π.χ., πίνακας, δομή, εγγραφή.

συνάρτηση (function) είναι μία αυτόνομη μονάδα κώδικα σχεδιασμένη να επιτελεί συγκεκριμένο έργο. Βοηθά στην αποφυγή της επανάληψης, στην αύξηση της επαναχρησιμοποίησης, στη βελτίωση της αναγνωσιμότητας του κώδικα, καθώς και στη βελτίωση της διαδικασίας συντήρησης.

συνδέτης (linker) ειδικό πρόγραμμα που συνδυάζει ανεξάρτητα μεταγλωττισμένα αρχεία αλλά και τις αναφορές τους σε βιβλιοθήκες με στόχο τη δημιουργία του εκτελέσιμου προγράμματος το οποίο αποθηκεύει στη βοηθητική μνήμη για μελλοντική χρήση.

σύνθετη πρόταση Σύνολο προτάσεων που περικλείονται μεταξύ ειδικών συμβόλων ({ και } για την C, begin και end για την Pascal) για προσδιορισμό της αρχής και του τέλους του συνόλου.

συντακτικό σφάλμα Σφάλμα που οφείλεται στην παραβίαση των συντακτικών κανόνων της γλώσσας. Αναγνωρίζεται και αναφέρεται από τον μεταγλωττιστή.

τελεστής (operator) ένα σύμβολο ή μία λέξη της γλώσσας προγραμματισμού, που αναπαριστά συγκεκριμένη διεργασία, η οποία εκτελείται πάνω σε ένα ή περισσότερα δεδομένα τα οποία καλούνται τελεστέοι (operands) και μπορούν να είναι μεταβλητές, σταθερές ή ακόμη κλήσεις συναρτήσεων.

τοπική μεταβλητή (local variable) μεταβλητή προσπελάσιμη μόνο από ένα συγκεκριμένο και σαφώς ορισμένο τμήμα κώδικα.

τριαδικός τελεστής (ternary operator) τελεστής που εφαρμόζεται σε τρεις τελεστέους π.χ., ο υποθετικός τελεστής ? : της C.

τυπικά ορίσματα (formal arguments) οι μεταβλητές που ορίζονται σε συναρτήσεις της C και δέχονται ως τιμές αυτές των πραγματικών ορισμάτων. Αποτελούν το εμφανές μέρος του μηχανισμού της C που υποστηρίζει το πέρασμα των εισόδων σε μια συνάρτηση.

τύπος δεδομένων (data type) είναι ένα σύνολο από αντικείμενα με κοινά χαρακτηριστικά τα οποία, συνήθως, εκφράζονται από ένα σύνολο πράξεων πάνω σε αυτά.

τύπος δείκτη Προσδιορίζει μεταβλητές οι οποίες χρησιμοποιούνται για την αποθήκευση διευθύνσεων της κύριας μνήμης του υπολογιστή.

υπολογισμός περιορισμένης έκτασης Μέθοδος υπολογισμού τιμής εκφράσεων. Έχει σαν αποτέλεσμα τον μη υπολογισμό του *y* α) στην έκφραση ***x and y*** όταν το *x* είναι ψευδές και β) στην έκφραση ***x or y*** όταν το *x* είναι αληθές.

υπορουτίνα Κατασκευή ανάλογη αυτής της συνάρτησης που οι γλώσσες προγραμματισμού χρησιμοποιούν για αναπαράσταση διεργασιών ή εφαρμογή της αφαιρετικότητας.

χρόνος εκτέλεσης (run–time) ο χρόνος που εκτελείται η εφαρμογή.

χρόνος μεταγλώττισης (compile–time) ο χρόνος που εκτελείται η μεταγλώττιση του πηγαίου κώδικα.