ΕΓΧΕΙΡΙΔΙΟ ΤΗΣ
ΓΛΩΣΣΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C

ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ – Ε.Μ.Π.

ΑΘΗΝΑ 2020

ΕΓΧΕΙΡΙΔΙΟ ΤΗΣ
ΓΛΩΣΣΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C

Βοήθημα για το μάθημα «**Εισαγωγή στην Πληροφορική και στον Προγραμματισμό**»

Έκδοση 2020.1

Παναγιώτης Παύλου, ΕΤΕΠ

ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ – Ε.Μ.Π.

ΑΘΗΝΑ 2020

Περιεχόμενα

Εισαγωγή στον προγραμματισμό

Σκοπός

Το παρόν κείμενο αποτελεί μία προσπάθεια να συγκεντρωθούν τα βασικά στοιχεία της γλώσσας προγραμματισμού C σε ένα συνοπτικό κείμενο. Κείμενο κατάλληλο όχι να διδαχθεί κάποιος προγραμματισμό ή τις λεπτομέρειες της γλώσσας C από αυτό, αλλά κατάλληλο να ανατρέχει όποτε χρειάζεται να δει συγκεντρωμένα τα στοιχεία της γλώσσας και να εντοπίσει όποιες λεπτομέρειες που του διαφεύγουν. Λειτουργεί δηλαδή, συμπληρωματικά προς τις σημειώσεις του μαθήματος.

Ο στόχος είναι, με τη βοήθειά του παρόντος, ο κάθε ένας και η κάθε μία, να ολοκληρώσει τις γνώσεις που θα αποκτήσει στο μάθημα. Γι'αυτό το λόγο, ελπίζουμε στη συνδρομή σας για τη συνεχή βελτίωσή τους. Επίσης σε κάθε σημείο στο οποίο δίνεται η ευκαιρία από τη διάταξή του κειμένου, υπάρχει χώρος για σημειώσεις ώστε να συμπληρώσει ό,τι κρίνει χρήσιμο ο κάθε ένας.

Τέλος, η πιο επικαιροποιημένη έκδοση του παρόντος κειμένου υπάρχει σε ηλεκτρονική μορφή, πάντα στο site του μαθήματος.

Αλγόριθμος

Γενικά

Ο **αλγόριθμος** είναι μια πεπερασμένη ακολουθία, συγκεκριμένων ενεργειών, εκτελούμενων σε πεπερασμένο χρόνο και αριθμό βημάτων. Ο σκοπός ενός αλγορίθμου είναι να τυποποιήσει την επίλυση ενός προβλήματος ή την εκτέλεση κάποιας εργασίας. Κάθε αλγόριθμος έχει **δεδομένα** στα οποία βασίζεται καθώς και **αποτελέσματα** τα οποία παράγει.

Αλγόριθμοι χρησιμοποιούνταν από την αρχαιότητα (τη Μεσοποταμία - 2500πΧ, την Αίγυπτο - 1500πΧ και την Ελλάδα - 300πΧ). Ενώ όμως η έννοια του αλγόριθμου είναι γενικότερη, χρησιμοποιείται πλέον σχεδόν αποκλειστικά στον χώρο της πληροφορικής, για να χαρακτηρίσει την περιγραφή μίας διαδικασίας προς στον υπολογιστή. Γι'αυτό το λόγο και οι αλγόριθμοι έχουν τύχει μελέτης από τα πρώτα βήματα της δημιουργίας των υπολογιστών. Μία έννοια που έχει μελετηθεί διεξοδικά είναι η **υπολογιστική πολυπλοκότητα** που σχετίζεται με την απόδοση ενός αλγορίθμου.

Ο κάθε αλγόριθμός μπορεί να εκφραστεί:

- πιο αφηρημένα με ένα **διάγραμμα ροής** ή με **ελεύθερο κείμενο**
- πιο συγκεκριμένα σε κάποια **γλώσσα προγραμματισμού**

Τα **βήματα** ενός αλγορίθμου συνήθως είναι ενός από τους παρακάτω τύπους:

1. Αρχικό και Τερματικό βήμα - που σηματοδοτεί το σημείο έναρξης και λήξης ενός αλγορίθμου
2. Απλό βήμα - που πραγματοποιεί μία μεμονωμένη "επεξεργασία"
3. Βήμα εισόδου/εξόδου - είσοδος δεδομένων ή έξοδος αποτελεσμάτων
4. Απόφαση/Διακλάδωση - όπου η ροή του αλγορίθμου μπορεί να ακολουθήσει μία από δύο εναλλακτικές διαδρομές.
5. Επιστροφή σε προηγούμενο βήμα/Επανάληψη - όπου κάποια προηγούμενα βήματα της διαδικασίας επαναλαμβάνονται.

Τα βήματα του αλγορίθμου εκτελούνται ένα κάθε φορά, το ένα μετά το άλλο. Το πρώτο βήμα καθορίζεται με κάποιο τρόπο, ανάλογα με το πως είναι εκφρασμένος. Για παράδειγμα στο διάγραμμα ροής υπάρχει ειδικός τύπος βήματος για την έναρξη, ενώ στη γλώσσα προγραμματισμού C υπάρχει η συνάρτηση main. Άρα η εκτέλεση ενός αλγορίθμου έχει μία ροή από την αρχή προς το τέλος ακολουθώντας τα βήματά του, η οποία ονομάζεται **ροή του αλγορίθμου** ή **ροή του προγράμματος**.

Ταξινόμηση αλγορίθμων

Η κατηγοριοποίηση των αλγορίθμων γίνεται με διάφορα κριτήρια, οι δύο πιο γενικές κατηγοριοποιήσεις είναι οι ακόλουθες.

Βάσει του αντικειμένου της επεξεργασίας

Οι κατηγορίες είναι πάρα πολλές, ενδεικτικές κατηγορίες αλγορίθμων είναι:

- Αναζήτησης τιμής σε σύνολο
- Ταξινόμησης τιμών
- Συγχώνευσης τιμών
- Αριθμητικών υπολογισμών
- Κειμένων
- Κρυπτογραφίας
- Συμπίεσης

Βάσει της πολυπλοκότητας

Εδώ η κατηγοριοποίηση αφορά το πως αυξάνεται ο χρόνος εκτέλεσης του αλγορίθμου βάσει της ποσότητας n της επεξεργαζόμενης πληροφορίας. Ακολουθεί μία λίστα με τις περιπτώσεις της κλιμάκωσης του χρόνου, βάσει του μεγέθους του προβλήματος. Δίπλα από την κάθε περίπτωση παρουσιάζεται και ο συμβολισμός της.

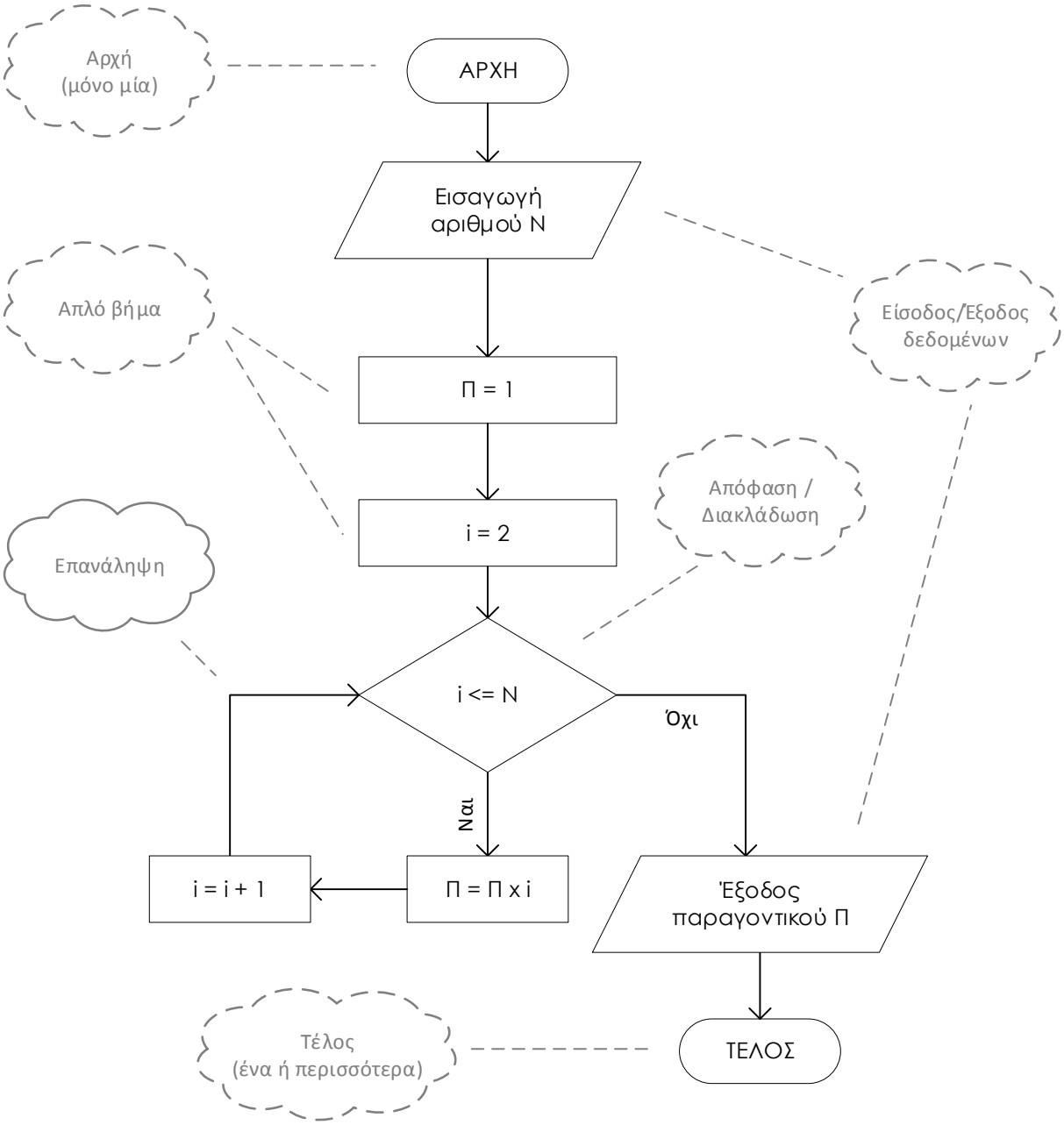
- Σταθερού χρόνου εκτέλεσης $O(1)$
  όταν το μέγεθος της πληροφορίας δεν παίζει ρόλο (π.χ. να βρεθεί το περιεχόμενο ενός στοιχείου πίνακα)
- Λογαριθμικού χρόνου $O(log\ n)$
  όπου ο χρόνος εκτέλεσης είναι ανάλογος με το μέγεθος της πληροφορίας (π.χ. Τεχνική της διχοτόμησης)
- Γραμμικού χρόνου $O(n)$
  όπου ο χρόνος εκτέλεσης είναι ανάλογος του μεγέθους (π.χ. να υπολογιστεί το άθροισμα των στοιχείων ενός πίνακα)
- Ημιλογαριθμικού χρόνου $O(n \cdot log\ n)$
  όπου ο χρόνος εκτέλεσης είναι ανάλογος με το μέγεθος πολλαπλασιασμένο με τον λογάριθμο του μεγέθους.
- Πολυωνυμικού χρόνου $O(n^k)$
  όπου ο χρόνος εκτέλεσης είναι ανάλογος με κάποια δύναμη του μεγέθους των δεδομένων (π.χ. Η απλή επεξεργασία ενός δισδιάστατου τετραγωνικού πίνακα είναι ανάλογη με το τετράγωνο του μεγέθους τους)
- Εκθετικού χρόνου $O(c^n)$
  όπου ο χρόνος εκτέλεσης είναι ανάλογος με κάποια σταθερά υψωμένη στο μεγεθος των δεδομένων.

ΣΗΜΕΙΩΣΕΙΣ

Διάγραμμα ροής

Το διάγραμμα ροής είναι ένας τρόπος να παρασταθεί γραφικά ένας αλγόριθμος. Το κάθε βήμα παριστάνεται με ένα γεωμετρικό σχήμα, ενώ το επόμενο βήμα υποδεικνύεται με ένα βέλος από το κάθε βήμα προς το επόμενο.

Παρακάτω φαίνεται ένα απλό διάγραμμα ροής του αλγορίθμου υπολογισμού του παραγοντικού. Τα σχήματα με τις διακεκομμένες είναι επεξηγηματικά και δεν αποτελούν μέρος του διαγράμματος ροής.



Η γλώσσα C

Ακολουθεί η ανάλυση των βασικών στοιχείων της γλώσσας C.

Δεδομένα

Η πληροφορία στους Η/Υ είναι αποθηκευμένη στη μνήμη τους **κωδικοποιημένη** με δυαδική μορφή, δηλαδή ως ακολουθία δύο ψηφίων 0 & 1. Όλα τα δεδομένα που χρειαζόμαστε να διαχειριστούμε μέσα σε έναν Η/Υ κωδικοποιούνται με αυτό τον τρόπο. Τα δεδομένα μπορεί να είναι αριθμοί (ακέραιοι ή δεκαδικοί), κείμενα, ημερομηνίες, χρώματα, ήχος, εικόνα, κλπ. Κάποια από αυτά είναι "απλά" και κάποια σύνθετα.

Εγγενή (Native) δεδομένα

Οι απλές πρωτογενείς πληροφορίες είναι αριθμητικές. Αυτές υποστηρίζονται - σε επίπεδο κυκλωμάτων - από τον ίδιο τον επεξεργαστή και τις ονομάζουμε **native** (εγγενείς). Με αυτό τον τρόπο παριστάνονται οι ακέραιοι και οι δεκαδικοί αριθμοί.

Οι υπόλοιπες πληροφορίες είναι σύνθετες και αποτελούνται από συνδυασμούς εγγενών (ή και σύνθετων) πληροφοριών.

Τύποι δεδομένων

Ο τύπος δεδομένων καθορίζει τον τρόπο με τον οποίο τα δυαδικά δεδομένα μεταφράζονται ή αντιστοιχούνται στην πληροφορία την οποία παριστάνουν. Αυτό σημαίνει ταυτόχρονα ότι τα ίδια δυαδικά δεδομένα όταν "μεταφραστούν" βάσει διαφορετικού τύπου δεδομένων, παριστάνουν διαφορετικές τιμές.

Για τα native δεδομένα η αντιστοίχηση δυαδικής πληροφορίας και τιμής καθορίζεται από την κατασκευή του επεξεργαστή.

Γενικά - όπως περιγράφεται και στο βιβλίο του μαθήματος - χρησιμοποιείται απλή αντιστοίχηση του δυαδικού συστήματος για τους **απρόσημους ακέραιους** ενώ για τους **προσημασμένους** το συμπλήρωμα ως προς δύο.

Αντίστοιχα για τους δεκαδικούς (**κινητής υποδιαστολής**) χρησιμοποιούνται τα πρότυπα της IEEE για την απλή (float) και τη διπλή (double) ακρίβεια.

ΣΗΜΕΙΩΣΕΙΣ

Τιμές

Η παράσταση των τιμών στη C εξαρτάται από τον τύπο δεδομένων που επιθυμεί ο προγραμματιστής να παραστήσει. Όπως παρουσιάζεται ακολούθως, οι ακέραιες και οι δεκαδικές τιμές παριστάνονται (προφανώς) με διαφορετικό τρόπο.

Ακέραιοι στο δεκαδικό σύστημα αρίθμησης

Οι **ακέραιες** τιμές στο **δεκαδικό** σύστημα παριστάνονται με διαδοχικά ψηφία που το πρώτο δεν μπορεί να είναι μηδέν. Μπροστά από τα αριθμητικά ψηφία μπορεί να υπάρχει πρόσημο (+ ή -). Για παράδειγμα:

```
1423     15972094854     −1523     −15682054     +54352     +777
```

Ακέραιοι στο οκταδικό σύστημα αρίθμησης

Οι **ακέραιες** τιμές στο **οκταδικό** σύστημα παριστάνονται αντίστοιχα με του δεκαδικού αλλά: (α) ξεκινούν με το ψηφίο 0 και (β) τα ψηφία που ακολουθούν θα πρέπει να είναι από 0-7. Οι τιμές αυτές μπορεί να είναι επίσης προσημασμένες. Για παράδειγμα:

```
0123     0128     −100     +777
```

Ακέραιοι στο δεκαεξαδικό σύστημα αρίθμησης

Οι **ακέραιες** τιμές στο **δεκαεξαδικό** σύστημα παριστάνονται ξεκινώντας την τιμή με το πρόθεμα 0x ενώ τα ψηφία που ακολουθούν μπορεί να είναι τα 0-9 ή A-F (ή a-f) όπου τα A - F αντιστοιχούν στις τιμές 10 - 15. Και πάλι οι τιμές αυτές μπορεί να είναι προσημασμένες. Για παράδειγμα:

```
0x100     0xA3F     −0x142     +0x777
```

ΣΗΜΕΙΩΣΕΙΣ

Ακέραιοι που παριστάνουν χαρακτήρες κειμένου

Οι **χαρακτήρες** κειμένου στη C παριστάνονται με μικρές **ακέραιες** τιμές. Ουσιαστικά αποτελούν μία αντιστοίχηση ακεραίων αριθμών σε χαρακτήρες κειμένου. Ο κάθε χαρακτήρας γίνεται κατανοητός στον υπολογιστή βάσει του κωδικού του. Η κωδικοποίηση που χρησιμοποιείται τυπικά στη C είναι η κωδικοποίηση ASCII (βλ. σχετικό παράρτημα). Όμως για διευκόλυνση, (όσο αφορά τουλάχιστον τους χαρακτήρες με κωδικό ASCII από 0 - 127), μπορεί να γραφεί με το σύμβολο του συγκεκριμένου χαρακτήρα (εφόσον αυτός είναι εκτυπώσιμος) μέσα σε μονά εισαγωγικά ώστε να μην είναι απαραίτητο να θυμάται ο προγραμματιστής την αριθμητική τιμή. Για παράδειγμα:

```
'a'    '-'    '8'    'Q'
```

που αντιστοιχούν στους κωδικούς, άρα και στις τιμές:

```
97    45    56    81
```

Μία άλλη γραφή χαρακτήρων είναι με τη χρήση του χαρακτήρα διαφυγής (escape character) `\` μέσα στα μονά εισαγωγικά, και γράφοντας στα δεξιά του κάποιον εκτυπώσιμο χαρακτήρα, του οποίου η σημασία αλλάζει. Αυτή η γραφή χρησιμοποιείται για να παραστήσει:

- Μη εκτυπώσιμους χαρακτήρες (δηλαδή χαρακτήρες που δεν φαίνονται), για παράδειγμα η αλλαγή γραμμής που παριστάνεται με το `'\n'`. Οι πιο συνηθισμένες ακολουθίες παρουσιάζονται στον πίνακα του παραρτήματος. Εάν η ακολουθία που γραφεί, δεν αντιστοιχεί σε κάποιον εκτυπώσιμο χαρακτήρα, τότε εμφανίζεται ο ίδιος ο χαρακτήρας που βρίσκεται δίπλα στον χαρακτήρα διαφυγής, π.χ. η ακολουθία `'\c'` είναι ο ίδιος ο χαρακτήρας `c`.
- Τον ίδιο τον χαρακτήρα του μονού εισαγωγικού που παριστάνεται με το `'\''`
- Τον ίδιο τον χαρακτήρα διαφυγής (escape character) που παριστάνεται με το `'\\'`
- Έναν χαρακτήρα με βάσει τον κωδικό του, εκφρασμένο στο οκταδικό σύστημα, για παράδειγμα `'\111'` που είναι στο δεκαδικό ο αριθμός 73 και βάσει του ASCII αντιστοιχεί στον χαρακτήρα `I` .

Προσοχή! Τα μονά εισαγωγικά δεν μπορεί ποτέ να είναι κενά (να μην έχουν ανάμεσά τους περιεχόμενο. Δηλαδή η γραφή `''` είναι **λάθος** και ο κώδικας που περιέχει ένα τέτοιο λάθος δεν μπορεί να γίνει compile. Ενώ το `' '` είναι σωστό καθώς έχει τον χαρακτήρα του κενού ανάμεσα στα εισαγωγικά.

ΣΗΜΕΙΩΣΕΙΣ

Πραγματικοί αριθμοί κινητής υποδιαστολής

Οι τιμές **κινητής υποδιαστολής** παριστάνονται μόνο στο δεκαδικό σύστημα αρίθμησης. Υπάρχουν δύο τρόποι παράστασης. Ο πρώτος αποτελεί γενίκευση της παράστασης των ακεραίων, όπου επιτρέπεται να υπάρχει και μία υποδιαστολή στο επιθυμητό σημείο. Η ύπαρξη της υποδιαστολής αρκεί για να καταστήσει τον αριθμό με τύπο δεδομένων κινητής υποδιαστολής. Χρειάζεται προσοχή στα εξής:

- Η υποδιαστολή παριστάνεται με . (τελεία) και όχι με , (κόμμα)
- Εφόσον υπάρχει η υποδιαστολή εάν τα στοιχεία του ακέραιου ή του δεκαδικού μέρους είναι όλα μηδενικά, τότε μπορούν να παραληφθούν.

Βέβαια δεν προτείνεται να παραλείπονται τα μηδενικά μέρη για καλύτερη αναγνωσιμότητα του κώδικα. Μπροστά από το πρώτο ψηφίο (που μπορεί να είναι αριθμητική ή η υποδιαστολή) μπορεί να υπάρχει πρόσημο (+ ή -). Παραδείγματα αυτής της γραφής είναι:

```
12.34    −43.21    0.123    +.123    123.    123.0
```

Ο δεύτερος τρόπος, είναι ο αντίστοιχος της εκθετικής ή επιστημονικής γραφής (scientific notation). Δηλαδή έναν αριθμό που θα γραφόταν ως $A \times 10^B$ τον γράφουμε ως **A**e**B** . Για παράδειγμα:

12.34×10^{13}	12.34e13
1.23×10^{-3}	1.23e−3
$-0.32478 \times 10^{1.5}$	−0.32478e1.5
$-0.5 \times 10^{-.5}$	−0.5e−.5

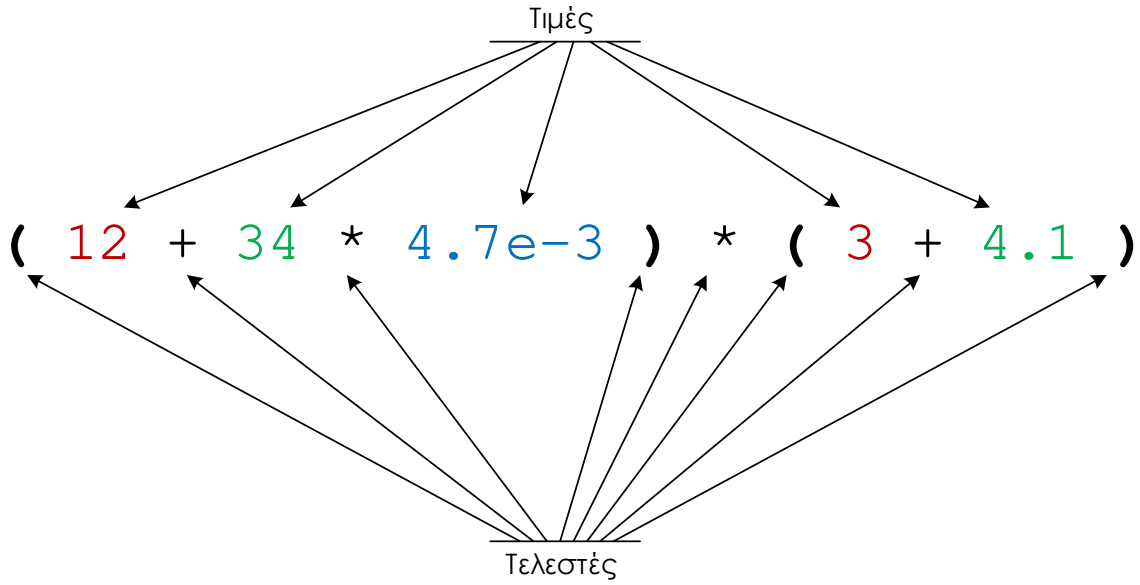
Προσοχή! Ανάμεσα στο σύμβολο **e**, και τους αριθμούς εκατέρωθεν, δεν πρέπει να έχουν κανένα κενό μεταξύ τους.

Προσοχή! Το **e** αυτό δεν έχει καμία σχέση με τη μαθηματική σταθερά e, τη βάση των φυσικών λογαρίθμων. Αντίθετα προέρχεται από τη λέξη exponent (=εκθέτης). Λειτουργεί ως διαχωριστικό της βάσης από τον εκθέτη.

ΣΗΜΕΙΩΣΕΙΣ

Τελεστές & παραστάσεις

Οι διάφορες τιμές συνδυάζονται για να προκύψουν **παραστάσεις**, οι οποίες αξιολογούνται-υπολογίζονται (evaluated) και προκύπτει μία τιμή που **αντικαθιστά** την παράσταση. Τα **σύμβολα** που χρησιμοποιούνται στις διάφορες παραστάσεις ονομάζονται **τελεστές**. Γνωστοί τελεστές από τα μαθηματικά είναι τα σύμβολα των τεσσάρων πράξεων και οι παρενθέσεις. Βέβαια εκτός από αυτούς, υπάρχουν πολύ περισσότεροι σε κάθε γλώσσα προγραμματισμού.



Στην παραπάνω παράσταση οι τιμές συνδυάζονται βάσει των τελεστών "κατά τα γνωστά", τα οποία όμως υπονοούν την ύπαρξη **προτεραιότητας** μεταξύ των τελεστών. Δηλαδή πρώτα υπολογίζω την παράσταση κάθε παρένθεσης, μετά γίνονται οι πολλαπλασιασμοί και οι διαιρέσεις και τέλος οι προσθέσεις και οι αφαιρέσεις. Η ιδέα της προτεραιότητας των τελεστών ή - όπως λέμε απλά - των πράξεων, επεκτείνεται και σε όλους τους τελεστές που έχει η γλώσσα προγραμματισμού.

Επίσης είναι σημαντικό να γίνει σαφές ότι: ο τελεστής δεν είναι απλά το σύμβολο, αλλά και η σχέση του με τις τιμές που αφορά. Δηλαδή το σύμβολο – είναι διαφορετικός τελεστής όταν χρησιμοποιείται για την αφαίρεση μιας τιμής από μια άλλη, και διαφορετικός τελεστής όταν λειτουργεί ως πρόσημο σε μία τιμή. Οι τιμές που σχετίζονται με τον τελεστή ονομάζονται **τελεσταίοι**.

Οι τελεστές, διακρίνονται μεταξύ τους βάσει της **πολλαπλότητας των τελεσταίων** τους οποίους εμπλέκουν. Με βάση αυτό οι τελεστές διακρίνονται σε μοναδιαίους (unary = 1 τελεσταίος π.χ. πρόσημα),δυαδικούς (binary = 2 τελεσταίοι π.χ. πρόσθεση) και τριαδικούς (ternary = 3 τελεσταίοι - υπάρχει μόνο ένας).

Άλλη διάκριση των τελεστών γίνεται βάσει της **λειτουργίας** που επιτελούν (αριθμητικοί, λογικοί, κλπ).

ΣΗΜΕΙΩΣΕΙΣ

Στο ακόλουθο πίνακα παρουσιάζεται η προτεραιότητα και η πολλαπλότητα των τελεσταίων.

Τελεστής	**Σχόλιο**	**Λειτουργία**	**Προτε-ραιότητα**	**Παράδειγμα χρήσης**	**Προσεταιρι-στικότητα**
++ --	Ως επίθεμα (μοναδιαίος/unary)	Αριθμητικός	1	x++ ή x--	Αριστερά προς τα δεξιά
()	Κλήση συνάρτησης			f()	
[]					
.					
->					
++ --	Ως πρόθεμα (μοναδιαίος/unary)	Αριθμητικός	2	++x ή –x	Δεξιά προς τα αριστερά
+	Πρόσημο (μοναδιαίος/unary)			+x	
-	Πρόσημο συμπλήρωμα προς 2 (μοναδιαίος/unary)			-x	
!		Λογικός		!x	
~	Συμπλήρωμα προς 1	Διαχείρισης Bits		~x	
(τύπος)	Type casting				
*****	Pointer dereference	Διαχείριση μνήμης		*x	
&	Address of			&x	
sizeof				sizeof x	
*** / %**	Πολλαπλασιασμός, διαίρεση , υπόλοιπο	Αριθμητικός	3	a * b ή a / b ή a % b	Αριστερά προς τα δεξιά
+ -	Πρόσθεση αφαίρεση		4		
<<	Δυαδική ολίσθηση αριστερά	Διαχείρισης Bits	5		
>>	Δυαδική ολίσθηση δεξιά				
<	Μικρότερος	Συσχέτισης	6		
<=					
>					
>=					
==			7		
!=					
&	AND bit προς bit	Διαχείρισης Bits	8		
^	XOR bit προς bit		9		
|	OR bit προς bit		10		
&&	Λογικό AND	Λογικός	11		
||	Λογικό OR		12		
?:	Τριαδικός (ternary)		13		Δεξιά προς τα αριστερά
= += -= *= /= %= <<= >>= &= ^= |=		Εκχώρησης	14		
,			15		Αριστερά προς τα δεξιά

Προσοχή! Σημειώστε ότι χρησιμοποιώντας παρενθέσεις, αλλάζει η σειρά υπολογισμού των παραστάσεων. Οι παρενθέσεις θα πρέπει να χρησιμοποιούνται άφοβα για να είναι ξεκάθαρη η σειρά υπολογισμού, όταν υπάρχουν αμφιβολίες για την προτεραιότητα των τελεστών. Η χρήση των παρενθέσεων δεν κοστίζει σε υπολογιστικό χρόνο καθώς απλά διαφοροποιούν τον κώδικα που παράγει ο compiler, δεν μεταφέρονται στον κώδικα της γλώσσας μηχανής ώστε να προκαλούν καθυστέρηση.

Μεταβλητές

Η έννοια της μεταβλητής είναι γνωστή από τα Μαθηματικά, αλλά στη γλώσσα C έχει κάποιες περισσότερες λεπτομέρειες που χρειάζονται προσοχή. Πιο συγκεκριμένα μία μεταβλητή αποτελείται από τα εξής:

- **Όνομα**, το οποίο υπακούει στους κανόνες των αναγνωριστικών της C (Identifiers) - βλ. την ενότητα των αναγνωριστικών
- **Τύπος δεδομένων**
- **Τιμή** – η οποία υπόκειται στους περιορισμούς του τύπου δεδομένων

Αυτό φαίνεται από μια πλήρη δήλωση μίας μεταβλητής όπως εδώ:

```
short int i = 1234;
```

Ξεκινά με τον τύπο δεδομένων (εδώ `short int`), ακολουθεί το όνομα (εδώ `i`) και στο τέλος είναι η τιμή. Η τιμή μεταφρασμένη βάσει του τύπου δεδομένου στο δυαδικό σύστημα αποθηκεύεται στη μνήμη `( 0000 0100 1101 0010 )`.

Η κάθε μεταβλητή μπορεί να χρησιμοποιηθεί στη θέση οποιασδήποτε τιμής μέσα σε μία παράσταση, ενώ το αποτέλεσμα μιας παράστασης μπορεί να αποθηκευτεί σε κάποια μεταβλητή, κατάλληλου τύπου δεδομένων.

ΣΗΜΕΙΩΣΕΙΣ

Τοπική (local)

Οι μεταβλητές δηλώνονται στα πλαίσια ενός block εντολών { ... }. Όταν η ροή του προγράμματος βγει εκτός των ορίων του block αυτού, τότε η μεταβλητή σταματά να υπάρχει. Η μνήμη στην οποία ήταν αποθηκευμένη απελευθερώνεται και δεν επιτρέπεται στον κώδικα πλέον να αναφερθεί σε αυτή. Αυτή η έκταση του προγράμματος στην οποία η μεταβλητή είναι προσβάσιμη, ονομάζεται εμβέλεια της μεταβλητής. Δεν επιτρέπεται να δηλωθούν δύο μεταβλητές με το ίδιο όνομα στο ίδιο block εντολών.

Στις περιπτώσεις που υπάρχουν εμφωλευμένα (nested) block εντολών, τότε οι μεταβλητές του εξωτερικού block είναι διαθέσιμες και στο εσωτερικό. Εάν δηλωθεί κάποια μεταβλητή με το ίδιο όνομα στο εσωτερικό block, τότε η εσώτερη μεταβλητή "κρύβει" την εξωτερική, μόνο για αυτό το block, από το σημείο της δήλωσης και κάτω. Αυτό φαίνεται στο παρακάτω κομμάτι κώδικα.

```
while (q > 0) {  //< - - - - - - - - - - - - - - - - Εξωτερικό block
   int A = 0;

   printf("Outer A : %d\n", A);       // 0

   if (A == 0) { //< - - - - - - - - - - - - - - - - Εσωτερικό block
      printf("Outer A : %d\n", A);    // 0

      int A = 1; //< - - - - - - - - - - - - - - - - Δήλωση μεταβλητής

      printf("Inner  A : %d\n", A);   // 1
   } //< - - - - - - - - - - - - - - - - - - - - - - - Τέλος εσωτερικού block

   printf("Outer A : %d\n", A);       // 0
}
```

Το ζήτημα της εμβέλειας βρίσκει ιδιαίτερη εφαρμογή στην περίπτωση των συναρτήσεων αφού το σώμα τους είναι πάντα block εντολών. Αυτό επιτρέπει τον ορισμό των μεταβλητών εντός της συνάρτησης με ονόματα της αρεσκείας μας χωρίς την ανησυχία μήπως συμπίπτουν τα ονόματα με άλλες μεταβλητές σε άλλες συναρτήσεις.

Δεδομένου ότι οι συναρτήσεις συνδέονται πάντα με ένα block εντολών, βάσει των παραπάνω βλέπουμε ότι δεν μπορούν να μοιράζονται μεταβλητές μεταξύ τους. Έτσι **τα δεδομένα μιας συνάρτησης είναι απομονωμένα από τα δεδομένα κάθε άλλης**. Λόγω αυτής της ιδιότητας των συναρτήσεων, καλό είναι να σημειώσουμε δύο περιπτώσεις που διαφέρουν.

Καθολική (global)

Παρότι αποτελεί πολύ κακή πρακτική στον προγραμματισμό, κάποιες φορές είναι χρήσιμο να υπάρχουν μεταβλητές κοινά διαθέσιμες σε όλες τις συναρτήσεις. Αυτές οι μεταβλητές ονομάζονται **global** (καθολικές). Η εμβέλειά τους είναι όλη η έκταση του αρχείου στο οποίο δηλώνονται. Η δήλωσή τους γίνεται με την ίδια σύνταξη όπως και οι τοπικές, αλλά διαφέρει είναι το σημείο του κώδικα στο οποίο δηλώνονται. Η δήλωση πρέπει να γίνεται "έξω" από όλες τις συναρτήσεις και συνήθως πριν τον ορισμό οποιασδήποτε συνάρτησης.

Στατική (static)

Επίσης οι μεταβλητές που έχουν δηλωθεί μέσα στη συνάρτηση, χάνονται μαζί με τις τιμές τους όταν ολοκληρωθεί η εκτέλεσή της. Εάν κάποια μεταβλητή θα πρέπει μετά την ολοκλήρωση της συνάρτησης να διατηρήσει τα περιεχόμενά της μέχρι την επόμενη κλήση της συνάρτησης τότε πρέπει να δηλωθεί ως **στατική**, προσθέτοντας τη "διευκρίνιση" static πριν τον ορισμό της. Για παράδειγμα για να "μετρά" μια συνάρτηση το πόσες φορές έχει κληθεί, γίνεται με τη χρήση μιας στατικής μεταβλητής.

```
void dummyFunction() {
  static int ctr = 0;
  ctr++;              // Μετρά πόσες φορές εκτελείται η συνάρτηση
}
```

Δηλαδή η στατική εμβέλεια, μοιάζει με την τοπική ως προς το που είναι διαθέσιμη η μεταβλητή, αλλά η τιμή διατηρείται από την κάθε κλήση στην επόμενη. **Η αρχική τιμή** που αναφέρεται (εδώ το 0) **δίνεται στη μεταβλητή** (εδώ `ctr`) **μόνο κατά την πρώτη εκτέλεση** αυτής της εντολής. Σε όλες τις επόμενες εκτελέσεις διατηρείται η προηγούμενη τιμή.

Σύνθετοι τύποι δεδομένων

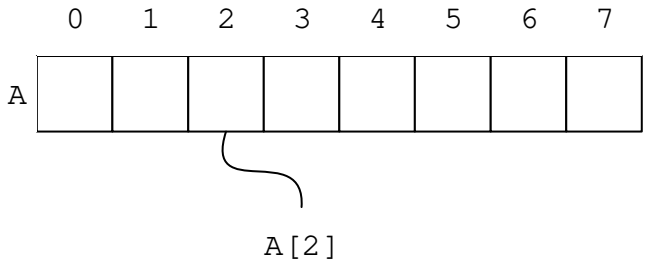
Οι σύνθετοι τύποι δεδομένων λειτουργούν σαν ομαδοποιήσεις "μεταβλητών". Οι "μεταβλητές" αυτές θα μπορούσαν να είναι οποιουδήποτε τύπου δεδομένων (εγγενών ή σύνθετων). Στην πραγματικότητα δεν είναι ακριβές να ονομάζονται μεταβλητές, αφού δεν έχουν αυτόνομο όνομα η κάθε μία.

Η ίδια η γλώσσα C έχει εγγενή υποστήριξη για τρεις μορφές σύνθετων τύπων δεδομένων:

- Τους πίνακες, όπου όλες οι εμπλεκόμενες "μεταβλητές" είναι του ίδιου τύπου δεδομένων και είναι αριθμημένες
- Τις συμβολοσειρές, που είναι πίνακες όπου τα στοιχεία τους είναι χαρακτήρες κειμένου
- Τις δομές, όπου οι εμπλεκόμενες μεταβλητές είναι ονοματισμένες και μπορεί να είναι διαφορετικού τύπου δεδομένων η κάθε μία

Πίνακες

Οι πίνακες, στη C, είναι μονοδιάστατες διατάξεις "μεταβλητών", κάτω από κοινό όνομα, που διακρίνονται βάσει ενός δείκτη ο οποίος παίρνει διαδοχικές ακέραιες τιμές από 0 μέχρι όσο χρειάζεται. Οπτικά παριστάνεται όπως παρακάτω:



Βλέπουμε ότι ο πίνακας συνολικά ονομάζεται όπως και οποιαδήποτε μεταβλητή, αλλά η διαφορά είναι ότι "αποτελείται" από πολλές "αριθμημένες" μεταβλητές μαζί (εδώ 8). Το ιδιαίτερο χαρακτηριστικό ενός πίνακα είναι ότι όλες οι μεταβλητές είναι υποχρεωτικά του ίδιου τύπου. Επίσης όταν δηλώνεται ένας πίνακας πρέπει να δίνεται μαζί και το μέγεθός του. Η **δήλωση** ενός πίνακα στη C γίνεται ως εξής:

```
int A[8];
```

Μια τέτοια δήλωση γίνεται όπως η δήλωση μιας απλής μεταβλητής ακολουθούμενη από το μέγεθος του πίνακα (δηλαδή το πλήθος των στοιχείων του) μέσα σε αγκύλες `[ ]`.

Ο τύπος δεδομένων του κάθε στοιχείου στο παραπάνω παράδειγμα είναι `int`, όμως θα μπορούσε να είναι οποιοσδήποτε τύπος δεδομένων, ακόμα και σύνθετος.

Όταν ορίζεται ένας πίνακας μπορούν να δοθούν αρχικές τιμές στα στοιχεία του όπως φαίνεται στο παρακάτω παράδειγμα:

```
int A[8] = { 1, 2, 4, 8, 10, 16, 24, 888 };
```

Δηλαδή μέσα σε άγκιστρα `{ }` γράφονται ένα προς ένα τα στοιχεία, χωρισμένα με κόμματα. Συνήθως το πλήθος των στοιχείων ταιριάζει με το μέγεθος του πίνακα που δηλώθηκε, όμως δεν είναι υποχρεωτικό.

ΣΗΜΕΙΩΣΕΙΣ

Αυτό μπορεί να αξιοποιηθεί με δύο τρόπους:

- Δεν είναι υποχρεωτικό να δοθούν όλα τα στοιχεία ενός πίνακα, αλλά μόνο όσα θέλει ο προγραμματιστής. Τα στοιχεία που θα δοθούν αντιστοιχούν στα πρώτα στοιχεία του, ξεκινώντας από τη θέση 0.
- Επίσης, όταν δίνονται όλα τα στοιχεία του πίνακα μέσα στα άγκιστρα μπορεί να παραληφθεί το μέγεθος του πίνακα. Έτσι εάν προστεθούν ή αφαιρεθούν στο μέλλον στοιχεία στα άγκιστρα, το μέγεθος του πίνακα θα προσαρμόζεται ανάλογα.

```
int A[8] = { 1, 2, 4 };
// Πίνακας 8 στοιχείων που
// τα 3 πρώτα είναι τα 1,2,4
```

```
int A[] = { 1, 2, 4 };
// Πίνακας 3 στοιχείων,
// των 1,2,4
```

Προσοχή! Οι πίνακες πρέπει να έχουν από την αρχή της ύπαρξής τους γνωστό μέγεθος/μήκος το οποίο στη συνέχεια δεν μπορεί να μεταβληθεί.

Αφού δηλωθεί ο πίνακας, η **αναφορά σε ένα από τα στοιχεία του πίνακα** γίνεται ως εξής:

```
A[i]
```

Όπου A είναι το όνομα του πίνακα και `i` η θέση του στοιχείου. Κατά τα άλλα, κάθε ένα στοιχείο του πίνακα όπως μία απλή μεταβλητή. Εάν πούμε ότι ο πίνακας `A` έχει μήκος `N`, τότε οι επιτρεπτοί δείκτες (δηλαδή οι τιμές του `i`) είναι από το `0`, έως και το `N-1`. Το μήκος ενός πίνακα δεν μπορεί να υπολογιστεί, θα πρέπει ο προγραμματιστής να φροντίσει να το "γνωρίζει" και να το λαμβάνει υπόψη του όταν και όπου χρειάζεται.

Προσοχή! Εάν γίνει αναφορά σε ένα στοιχείο του πίνακα "έξω από αυτόν", είτε με αρνητική (!) τιμή του δείκτη, είτε με τιμή μεγαλύτερη ή ίση με το μέγεθός του, τότε η συμπεριφορά του προγράμματος είναι ακαθόριστη. Μπορεί να σταματήσει εντελώς η εκτέλεσή του προγράμματος, ή να προκύψει κάποια τιμή άσχετη με τη λογική του προγράμματος ή να αλλάξουν οι τιμές από άλλες μεταβλητές. Όταν προκύπτουν τέτοια σφάλματα είναι πολύ δύσκολο να εντοπιστούν.

ΣΗΜΕΙΩΣΕΙΣ

Πολυδιάστατοι πίνακες

Καθώς αναφέρθηκε, τα στοιχεία των πινάκων είναι όλα του ίδιου τύπου, αλλά δεν υπάρχει περιορισμός στο ποιος τύπος είναι αυτός. Ενδιαφέρον παρουσιάζει η ειδική περίπτωση, όπου τα στοιχεία ενός πίνακα είναι πίνακες. Για παράδειγμα ένας απλός πίνακας πραγματικών αριθμών θα δηλωνόταν ως:

```
double x[3];
```

Άρα ένας πίνακας πινάκων θα δηλωνόταν ως:

```
double x[3][4];
```

και θα λειτουργούσε ως ένας πίνακας δύο διαστάσεων. Θα πρέπει να θεωρηθεί ότι η αριστερή αρίθμηση είναι οι γραμμές και η δεξιά οι στήλες. Η αποθήκευση των πινάκων στη C γίνεται κατά γραμμές. Οι πίνακες 2 διαστάσεων έχουν τα στοιχεία τους διαδοχικά αποθηκευμένα στη μνήμη. Μετά την 1η γραμμή ακολουθεί η 2η, κατόπιν τη 3η κοκ. Άρα οπτικά:

	0	1	2	3	4	5	6	7	8	9	10	11
x	x [0][0]	x [0][1]	x [0][2]	x [0][3]	x [1][0]	x [1][1]	x [1][2]	x [1][3]	x [2][0]	x [2][1]	x [2][2]	x [2][3]

Κείμενα

Τα κείμενα αποτελούν μία ειδική περίπτωση πινάκων, όπου τα στοιχεία του πίνακα είναι τύπου `char`, δηλαδή μπορούν να αποθηκεύουν έναν χαρακτήρα κειμένου το κάθε ένα. Στη C θα τα δείτε με την ονομασία strings ή συμβολοσειρές.

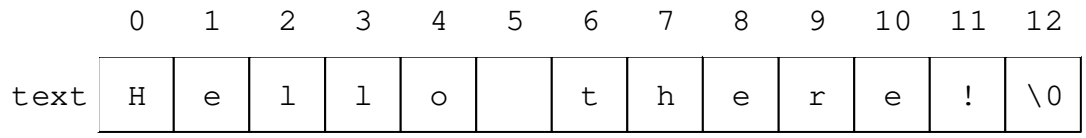
Η ιδιαιτερότητα που έχουν τα κείμενα στη C έγκειται σε δύο σημεία. Το πρώτο είναι ότι το κείμενο ως πίνακας, μπορεί να δοθεί και γραμμένο μέσα σε διπλά εισαγωγικά, κάνοντας πιο ευανάγνωστο και κατανοητό τον κώδικα. Π.χ.

```
char text[] = "Hello there!";
```

που είναι ισοδύναμο με το:

```
char text[] = {'H','e','l','l','o',' ','t','h','e','r','e','!','\0'};
```

Αυτό παριστάνεται οπτικά ως ακολούθως:



Δηλαδή είναι ένας πίνακας με ένα χαρακτήρα σε κάθε στοιχείο του, στο στοιχείο στη θέση 5 (δηλαδή το 6ο στοιχείο) είναι αποθηκευμένο το κενό (space). Επίσης στο τέλος είναι αποθηκευμένος ένας χαρακτήρας με κωδικό 0 που δηλώνει ότι εκεί είναι το τέλος του κειμένου. Έτσι ο πίνακας έχει μέγεθος/μήκος 13 ενώ το μήκος του κειμένου είναι 12 χαρακτήρες.

Το δεύτερο είναι ακριβώς αυτό που αναφέρθηκε παραπάνω, ότι δεν χρειάζεται να είναι γνωστό το μήκος του πίνακα, αλλά αντίθετα πρέπει μετά τον τελευταίο χαρακτήρα του κειμένου να υπάρχει τουλάχιστον ακόμη ένα στοιχείο που να έχει την τιμή 0 (όχι τον κωδικό του χαρακτήρα '0'), το οποίο συμβολίζεται και με το '\0'. Με αυτό τον τρόπο ο πίνακας χρειάζεται να έχει ένα στοιχείο περισσότερο, αλλά αποφεύγεται η απαίτηση να είναι γνωστό εκ των προτέρων το μήκος του.

Αντίστροφα εάν υπάρχει κάποιος πίνακας αρκετά μεγάλος ώστε να μπορεί να αποθηκεύσει πολλούς χαρακτήρες, τότε δεν είναι απαραίτητο να χρησιμοποιείται όλη η έκτασή του (όλα τα στοιχεία του) για το κείμενο, αλλά μόνο όσα χρειάζονται, από την αρχή του, μέχρι να βρεθεί στοιχείο με κωδικό 0. Δηλαδή εναλλακτικά θα μπορούσε να είναι ως εξής:

```
char text[15] = "Hello there!";
```

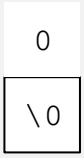
που αντιστοιχεί οπτικά στο:

Δηλαδή τα δύο τελευταία στοιχεία, τα οποία υπάρχουν μεν, αλλά δεν είναι αναγκαία για την αποθήκευση του κειμένου, συνήθως έχουν ως περιεχόμενο τον κωδικό 0, είτε περιεχόμενο που δεν είναι δυνατόν να προβλεφθεί. Αυτό εξαρτάται από τον compiler.

Εδώ ο πίνακας έχει μέγεθος 15 χαρακτήρες, αλλά το μήκος του κειμένου (χωρίς να μετράμε τον μηδενικό/τερματικό χαρακτήρα) είναι 12 χαρακτήρες.

Ανεξαρτήτως του τρόπου με τον οποίο δηλώθηκε η `text`, οπουδήποτε χρησιμοποιηθεί ως κείμενο (π.χ. εάν εκτυπωθεί στην οθόνη το περιεχόμενο του πίνακα `text` με χρήση της `printf` ως κείμενο (δηλ. `%s`)), δεν θα υπάρχει διαφοροποίηση στο αποτέλεσμα.

Προσοχή! Σε αντίθεση με τα μονά εισαγωγικά, στα διπλά εισαγωγικά επιτρέπεται και έχει νόημα να δηλώσουμε ένα κείμενο ως **""** δηλαδή ένα κείμενο χωρίς κανένα χαρακτήρα. Αυτό το κείμενο παρουσιάζεται ως:



και ο πίνακας έχει μέγεθος 1 και το κείμενο μήκος 0.

Προσοχή! Τα κείμενα μέσα στα διπλά εισαγωγικά, πρέπει να ξεκινούν και να τελειώνουν στην ίδια γραμμή. **Δεν επιτρέπονται** κείμενα όπως το ακόλουθο, οπότε και αυτός ο κώδικας δεν κάνει compile:

```
char text[] = "this is
               a test";
```

ΣΗΜΕΙΩΣΕΙΣ

Δομές (struct)

Οι δομές είναι σύνθετοι τύποι δεδομένων, που ομαδοποιούν ονοματισμένες μεταβλητές ίδιου ή διαφορετικών τύπων δεδομένων, απλών ή σύνθετων. Οι "εσωτερικές" αυτές μεταβλητές ονομάζονται και πεδία (fields) ή μέλη (members). Η κάθε δομή είναι και αυτή ονοματισμένη. Επειδή συνήθως η δομή "ομαδοποιεί" μεταβλητές που έχουν κάποια σχέση μεταξύ τους (π.χ. περιγράφουν ένα φυσικό αντικείμενο), καλό είναι το όνομά της συμπίπτει με το όνομα του αντικειμένου. Ακολουθεί ένα παράδειγμα ορισμού μιας δομής:

```
struct Particle {
  double x;
  double y;
  double z;
  double mass;
  char label[10];
}
```

Εδώ λοιπόν το όνομα της δομής είναι Partice, αφού περιγράφει ένα σωματίδιο στον χώρο. Τα πεδία της δομής είναι:

- τα x,y,z (τύπου double) που αντιστοιχούν στη θέση του σωματιδίου στον χώρο
- mass (τύπου double) που αντιστοιχεί στη μάζα του σωματιδίου και τέλος
- label που είναι ένα κείμενο και είναι η "λεκτική ετικέτα" που έχει το σωματίδιο

Μέχρι αυτό το σημείο έχει οριστεί η δομή, αλλά στην πραγματικότητα δεν έχει καταναλωθεί καθόλου αποθηκευτικός χώρος στον Η/Υ, καθώς ενώ έχει οριστεί η "έννοια του σωματιδίου", δεν έχει δηλωθεί κανένα σωματίδιο. Η δήλωση μιας μεταβλητής με όνομα A, η οποία "είναι τύπου σωματιδίου" γίνεται ως ακολούθως:

```
struct Particle A;
```

Τώρα πλέον μπορούν να χρησιμοποιηθούν τα μέλη της δομής για το σωματίδιο A, ως εξής:

```
A.mass = 0.0;
A.x = A.y = A.z = 0.0;
```

Δηλαδή είναι σαν να υπάρχουν διαθέσιμες νέες μεταβλητές που το όνομά τους αποτελείται από δύο μέρη. Το πρώτο μέρος είναι το όνομα της μεταβλητής (τύπου σωματιδίου) `A` και το δεύτερο είναι το όνομα του μέλους (π.χ. `mass`), ενωμένα με μία τελεία.

ΣΗΜΕΙΩΣΕΙΣ

Άρα συνολικά εμπλέκονται τρεις ονομασίες:

- Η ονομασία της ίδιας της δομής (`struct Particle`)
- Η ονομασία της μεταβλητής (του τύπου της δομής) `A`
- Η ονομασία του πεδίου π.χ. `mass`

Επειδή είναι άβολο να μεταφέρεται η λέξη `struct` συνεχώς, εφόσον η δομή δηλωθεί με τη χρήση της `typedef`, μπορεί κάποιος να χρησιμοποιεί κατευθείαν το όνομα αυτό ως τύπο δεδομένων:

```
typedef struct _particle {
      double x;
      double y;
      double z;
      double mass;
      char label[10];
} Particle;
```

Δηλαδή επιλέγεται ένα διαφορετικό όνομα για τη δομή (π.χ. το `_particle`) το οποίο δεν χρειάζεται να χρησιμοποιηθεί σε άλλο σημείο του κώδικα, και ορίζεται η λέξη `Particle` να ισοδυναμεί με το `struct _particle`. Άρα πλέον ένα σωματίδιο μπορεί να ορίζεται απλά ως:

```
Particle A;
A.mass = 100.0;
```

Ή εάν για κάποιο λόγο είναι επιθυμητό να χρησιμοποιηθεί το όνομα της δομής, ισοδύναμα γράφεται:

```
struct _particle A;
A.mass = 100.0;
```

Προσοχή! Ενώ κάτι τέτοιο δεν είναι απαραίτητο αποτελεί πολύ καλή πρακτική στο προγραμματισμό, για κάθε δομή που ορίζετε να δημιουργείτε και τουλάχιστον μία συνάρτηση που να δίνει αρχικές τιμές σε όλα τα μέλη της. Ακόμα καλύτερα, η συνάρτηση θα μπορούσε να δημιουργεί τη δομή με τη χρήση της `malloc` και να την επιστρέφει. Η συγκεκριμένη τεχνική αποτελεί κοινό τόπο στον αντικειμενοστραφή προγραμματισμό (object oriented programming) και αυτή η κατηγορία συναρτήσεων ονομάζεται constructors. Αν και δεν είναι προφανές τέτοια συμπεριφορά έχει και η συνάρτηση `fopen` η οποία ανοίγει ένα αρχείο στον δίσκο.

Αναγνωριστικά/Identifiers

Αναγνωριστικά ονομάζουμε όλα τα λεκτικά/ονόματα που επιτρέπεται να χρησιμοποιεί ο προγραμματιστής σε ένα πρόγραμμα C. Αυτά μπορούν να χρησιμοποιούνται ως:

- Ονόματα σταθερών
- Ονόματα μεταβλητών
- Ονόματα συναρτήσεων
- Ονόματα τύπων δεομένων

Τα αναγνωριστικά αποτελούνται από όσους χαρακτήρες θέλει ο προγραμματιστής και οι χαρακτήρες που επιτρέπονται είναι οι λατινικοί χαρακτήρες (πεζοί `a-z` και κεφαλαίοι `A-Z`), η κάτω παύλα (underscore _) και (εκτός από τον πρώτο χαρακτήρα της ονομασίας) αριθμητικοί χαρακτήρες (`0-9`). Επίσης οι ονομασίες είναι "case sensitive", δηλαδή οι πεζοί και οι κεφαλαίοι χαρακτήρες, είναι διαφορετικοί. Άρα τα αναγνωριστικά `someVariableName` και `somevariablename` είναι διαφορετικά. Επίσης υπάρχουν δεσμευμένες λέξεις της C που ενώ ικανοποιούν όλες τις παραπάνω συνθήκες δεν επιτρέπεται να χρησιμοποιηθούν ως αναγνωριστικά. Οι δεσμευμένες λέξεις - ανάλογα και την έκδοση της C - συνήθως είναι οι ακόλουθες:

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

Τα αναγνωριστικά αυτά, βοηθά πολύ να είναι περιγραφικά και αυτό συχνά σημαίνει να αποτελούνται από περισσότερες από μία λέξεις (π.χ. μία μεταβλητή που κρατά το πλήθος των αποτυχημένων προσπαθειών ενός σύνδεσης χρήστη σε ένα σύστημα θα μπορούσε να ονομαστεί *User Attempt Count*).

Η απουσία των κενών και όλων των ειδικών χαρακτήρων από τις ονομασίες (πλην του underscore) περιορίζει τις επιλογές για το πώς διαχωρίζονται οι λέξεις σε ένα αναγνωριστικό. Έτσι πρέπει να χρησιμοποιηθεί άλλος τρόπος για να ξεχωρίζουν οι λέξεις. Οι λύσεις που χρησιμοποιούνται συνήθως για να γραφεί ως αναγνωριστικό μία φράση είναι συνήθως (βάσει του παραπάνω παραδείγματος):

- `userAttemptCount` - που ονομάζεται *camelCase* επειδή τα κεφαλαία αρχικά των ενδιάμεσων λέξεων σχηματίζουν καμπούρες καμήλας. Αυτό το στυλ ενδείκνυται για τις ονομασίες των **μεταβλητών** και προαιρετικά των **συναρτήσεων**.
- `UserAttemptCount` - που ονομάζεται *PascalCase* επειδή χρησιμοποιούταν αρχικά στη γλώσσα προγραμματισμού Pascal. Αυτό το στυλ χρησιμοποιείται για τις ονομασίες νέων **τύπων δεδομένων** (`struct` & `typedef`) και ως εναλλακτική επιλογή για τα ονόματα **συναρτήσεων**.
- `user_attempt_count` - που ονομάζεται και *snake_case* επειδή είναι όλο ιδιαίτερα κοντά στο… έδαφος. Χρησιμοποιείται σπανιότερα πλέον και κυρίως για μεταβλητές στη C.
- `USER_ATTEMPT_COUNT` - που ονομάζεται και *ALL_CAPS* επειδή όλοι οι χαρακτήρες είναι γραμμένοι στα κεφαλαία. Αυτή η γραφή χρησιμοποιείται συνήθως για να ονοματιστούν **σταθερές** ποσότητες.

Συναρτήσεις

Συναρτήσεις ονομάζονται τα αυτοτελή τμήματα κώδικα που έχουν σκοπό να επιτελέσουν συγκεκριμένη λειτουργία. Τα τμήματα αυτά είναι ονοματισμένα (με ένα αναγνωριστικό) το οποίο συνήθως δηλώνει τον σκοπό του κώδικα που περιέχει η συνάρτηση. Δέχονται ως παραμέτρους (parameters) μεταβλητές που αποτελούν "είσοδο" στη συνάρτηση και επιστρέφουν μία μόνο τιμή ως αποτέλεσμα της εκτέλεσής της. Οι παράμετροι αλλά και το αποτέλεσμα θα πρέπει να έχουν εκ των προτέρων γνωστούς τύπους δεδομένων.

Με τις συναρτήσεις ο κώδικας μπορεί και χωρίζεται σε επαναχρησιμοποιούμενες λειτουργικές ενότητες, κάνοντας πιο κατανοητό τον σκοπό του κάθε αντίστοιχου τμήματος κώδικα και πιο ευανάγνωστο συνολικά το πρόγραμμα.

Ορισμός

Μία τυπική συνάρτηση όπως η ***squareOf(x) = x²*** ορίζεται (**defined**) στη C ως ακολούθως:

```
double squareOf(double x) {
      return x * x;
}
```

Δηλαδή γράφονται:

- ο τύπος δεδομένων του αποτελέσματός της
- το όνομά της
- οι παράμετροι με τον τύπο τους μέσα σε παρενθέσεις
- ο κώδικας της συνάρτησης μέσα σε άγκιστρα `{ }`.

Σε κάποιο σημείο στο εσωτερικό του κώδικα, πρέπει να υπάρχει τουλάχιστον μία εντολή `return` η οποία ακολουθείται από την τιμή του αποτελέσματος της συνάρτησης και η οποία όταν εκτελεστεί τερματίζει και την εκτέλεση της συνάρτησης.

Μετά τον ορισμό μιας συνάρτησης, μπορεί να κληθεί από οποιοδήποτε επιτρεπτό σημείο του κώδικα. Παράδειγμα κλήσης της παραπάνω συνάρτησης είναι:

```
double squareOfFive = squareOf(5.0);
```

Όταν έρθει η στιγμή να εκτελεστεί η παραπάνω γραμμή του κώδικα, τότε:

- οι τιμές μέσα στις παρενθέσεις (εδώ μόνο το 5.0) - οι οποίες ονομάζονται ορίσματα ή arguments - αντιγράφονται στις παραμέτρους (parameters) της συνάρτησης, οι οποίες παράμετροι για τη συνάρτηση λειτουργούν ως τοπικές μεταβλητές.
- εκτελείται ο κώδικας της συνάρτησης μέχρι να βρεθεί η εντολή return ή να συναντηθεί το τελικό άγκιστρο της συνάρτησης. Προσέξτε ότι ιδανικά πρέπει να υπάρχει τουλάχιστον μία εντολή return σε κάθε συνάρτηση.

- το αποτέλεσμα που έχει επιστρέψει η εντολή return, αντιγράφεται στη θέση της συνάρτησης, στη γραμμή από την οποία κλήθηκε και συνεχίζεται η εκτέλεση αυτής της γραμμής.

Προσοχή! Τόσο η δήλωση, όσο και ο ορισμός μιας συνάρτησης εξυπακούεται ότι δεν μπορούν να γίνουν στο εσωτερικό μιας άλλης συνάρτησης.

Ορίσματα και Παράμετροι

Στο πρώτο βήμα της διαδικασίας κλήσης μιας συνάρτησης, είναι σημαντική η διάκριση μεταξύ ορισμάτων και παραμέτρων.

Ορίσματα (arguments) ονομάζονται οι τιμές που δίνονται στη συνάρτηση (μέσα στις παρενθέσεις) κατά την κλήση της.

Παράμετροι (parameters) ονομάζονται οι μεταβλητές οι οποίες υποδέχονται τις παραπάνω τιμές και γράφονται πάλι μέσα σε παρενθέσεις στον ορισμό της συνάρτησης.

Από αυτό και μόνο, γίνεται αντιληπτό ότι η κάθε συνάρτηση κατά την εκτέλεσή της χειρίζεται αντίγραφα τιμών. Άρα σε έναν κώδικα όπως ο ακόλουθος

```
someUnknownFunction ( aVariable, anotherVariable );
```

δεν είναι ποτέ δυνατόν να αλλαχθεί η τιμή της μεταβλητής `aVariable` ή της `anotherVariable` κατά την κλήση της συνάρτησης `someUnknownFunction`, ανεξαρτήτως του κώδικα τον οποίο περιέχει η συνάρτηση.

Ο τρόπος κλήσης των συναρτήσεων στη C είναι μόνο αυτός και ονομάζεται **κλήση με τιμή** (**call by value**).

ΣΗΜΕΙΩΣΕΙΣ

Δήλωση

Ένα άλλο σημείο που αναφέρθηκε προηγουμένως είναι ότι μετά τον ορισμό της συνάρτησης, αυτή μπορεί να χρησιμοποιηθεί όπου είναι επιθυμητό. Αυτό δημιουργεί έναν περιορισμό σε κάποιες περιπτώσεις.

Μία περίπτωση είναι όταν χρειαστεί να οριστούν δύο συναρτήσεις Α και Β μέσα στον κώδικα των οποίων, η κάθε μία καλεί την άλλη!

```
int A() {
   ...
   B();   // Ωχ! Η Β δεν είναι ορισμένη
   ...
}

int B() {
   ...
   A();   // Ok είναι ήδη ορισμένη
   ...
}
```

Μία δεύτερη περίπτωση προκύπτει όταν αναλύει κάποιος ένα πρόβλημα. Η ανάλυση ξεκινά με τη γενική επίλυση του "σπάζοντάς" το σε επιμέρους προβλήματα. Κάθε ένα από αυτά συνήθως αντιστοιχεί σε μία συνάρτηση, η οποία γράφεται στη συνέχεια ώστε να λυθούν τα προβλήματα αυτά. Βάσει του παραπάνω περιορισμού θα έπρεπε να γραφούν πρώτα οι συναρτήσεις που λύνουν τα επιμέρους προβλήματα και στο τέλος η συνάρτηση που λύνει το κύριο πρόβλημα. Αυτό μπορεί να μπερδέψει εύκολα όποιον διαβάζει τον κώδικα.

```
void mainProblem() { ...
    A = solveA(); // Δεν μπορώ να καλέσω την solveA() ακόμα!
    B = solveB(); // Δεν μπορώ να καλέσω την solveB() ακόμα!
...}

int solveA() { ... }
int solveB() { ... }
```

Για να λυθεί αυτό το ζήτημα, η C προβλέπει να μπορεί σε οποιοδήποτε σημείο του κώδικα, να δηλωθεί (**declare**) μία συνάρτηση. Η γραφή είναι η ίδια με του ορισμού (**definition**), αλλά λείπει εντελώς ο κώδικας της και φυσικά να τελειώνει με ένα ερωτηματικό (Αγγλική άνω τελεία). Για παράδειγμα:

```
double squareOf(double x);
```

Μετά τη δήλωση της συνάρτησης, αυτή μπορεί να κληθεί κανονικά. Αρκεί σε κάποιο, οποιοδήποτε, σημείο ακολούθως να δοθεί και ο ορισμός (definition) της.

Προσοχή! Η δήλωση, όπως και ο ορισμός, δεν μπορεί να γίνει στο εσωτερικό μιας άλλης συνάρτησης.

Αναδρομή

Η **αναδρομή** είναι μία προγραμματιστική τεχνική όπου μία συνάρτηση, μέσα στο σώμα των εντολών της, να καλεί τον ίδιο της "τον εαυτό". Αυτή η τεχνική ονομάζεται αναδρομή (recursion) και χρησιμοποιείται συχνά.

Σε κάθε περίπτωση η αναδρομή μπορεί να γραφεί και ως επαναληπτική διαδικασία. Όμως είναι αρκετά προβλήματα, στα οποία η λύση τους περιγράφεται πολύ ευκολότερα ανάγοντας το πρόβλημα στην επίλυση ενός προβλήματος του ίδιο τύπου αλλά μικρότερης τάξης. Σε αυτά ταιριάζει η χρήση της αναδρομικής κλήσης μιας συνάρτησης.

Προσοχή! Κάθε φορά που η συνάρτηση καλεί τον εαυτό της, από την πλευρά του υπολογιστή είναι σαν να καλείται μία οποιαδήποτε συνάρτηση. Άρα οι παράμετροι και όλες οι τοπικές μεταβλητές, δημιουργούνται ως ένα νέο αντίγραφο για την νέα εκτέλεση. Έτσι αυξάνεται η μνήμη που καταναλώνεται, γι'αυτό η αλόγιστη χρήση της αναδρομής μπορεί να είναι πολύ σπάταλη σε μνήμη, χωρίς όφελος.

Για παράδειγμα η παρακάτω συνάρτηση:

```
void showForward(int x) {
    if (x < 0) {
        return;
    }

    showForward(x-1);

    printf("%i\n", x);

    return;
}
```

Καλεί συνεχώς τον εαυτό της μέχρι να φτάσει η παράμετρος να έχει αρνητική τιμή. Κατόπιν εκτυπώνει την τιμή αυτή και επιστρέφει.

Προσοχή! Η αναδρομή για να είναι εφικτή θα πρέπει κάποια στιγμή να τερματίζεται, αλλιώς η συνάρτηση θα καλεί αενάως τον εαυτό της μέχρι να τελειώσει η μνήμη του υπολογιστή, η μνήμη του stack ή να φτάσει η εκτέλεση σε κάποιο μέγιστο βάθος αναδρομής. Σε κάθε περίπτωση η εκτέλεση του προγράμματος δεν ολοκληρώνεται.

ΣΗΜΕΙΩΣΕΙΣ

Έλεγχος Ροής

Αποφάσεις

Η εντολή **if…else…**

Σκοπός

Η εντολή if επιτρέπει την επιλεκτική εκτέλεση μιας ομάδας (block) εντολών, βάσει της αλήθειας μίας συνθήκης (test expression). Προαιρετικά – με τη χρήση της else - μπορεί να οριστεί και ένα εναλλακτικό block εντολών το οποίο θα εκτελεστεί εφόσον η συνθήκη είναι ψευδής.

Σύνταξη

Η σύνταξη της εντολής `if` στη C είναι:

```
if (test expression) {
    // εντολές που εκτελούνται εάν η συνθήκη είναι αληθής
}
```

Η σύνταξη της εντολής `if` με `else` στη C είναι:

```
if (testExpression) {
    // εντολές που εκτελούνται εάν η συνθήκη είναι αληθής
} else {
    // εντολές που εκτελούνται εάν η συνθήκη είναι ψευδής
}
```

ή για αλλεπάλληλες `if else`

```
if (testExpression1) {
    // εντολές που εκτελούνται εάν η συνθήκη1 είναι αληθής
} else if (testExpression2){
    // εντολές που εκτελούνται εάν η συνθήκη2 είναι αληθής
    // ενώ η testExpression1 είναι ψευδής
} else if (testExpression3){
    // εντολές που εκτελούνται εάν η συνθήκη3 είναι αληθής
    // ενώ η testExpression1 και η testExpression2 είναι ψευδείς
} else {
    // εντολές που εκτελούνται εάν οι παραπάνω συνθήκες είναι ψευδείς
}
```

Παραδείγματα

Απόλυτη τιμή του x

```
if (x<0) {
      x=-x;
}
```

Τοποθέτηση της μικρότερης τιμής στο x και της μεγαλύτερης στο y

```
if (x>y){
      int t = x;
      x = y;
      y = t;
}
```

Μεγαλύτερος εκ των x και y

```
if (x>y){
      max_xy = x;
} else {
      max_xy = y;
}
```

Ρίζες τριωνύμου ax^2+bx+c

```
double diakrinousa = b*b - 4.0*c;
if (diakrinousa < 0.0){
    printf("No real roots");
} else {
    x1 = (-b + sqrt(diakrinousa))/ 2.0 ;
    x2 = (-b - sqrt(diakrinousa))/ 2.0 ;
}
```

Έλεγχος ακεραίου για το αν είναι ζυγός ή περιττός

```
if (number%2 == 0) {
    printf("%d is an even integer.",number);
} else {
    printf("%d is an odd integer.",number);
}
```

Έλεγχος έτους αν είναι δίσεκτο

```
if (year%400 == 0) {
    // Εάν διαιρείται με το 400 είναι δίσεκτο
    isLeap=TRUE;
} else if(year%100 == 0) {
    // Εάν διαιρείται με το 100 αλλά όχι με το 400 δεν είναι δίσεκτο
    isLeap=FALSE;
} else if(year%4 == 0) {
    // Εάν διαιρείται με το 4 αλλά ούτε με το 400 ούτε με το 100
    // είναι δίσεκτο
    isLeap=TRUE;
} else {
    // Εάν δεν διαιρείται με το 4, το 100, ούτε το 400, δεν είναι δίσεκτο
    isLeap=FALSE;
}
```

Η εντολή **switch...case...default**

Σκοπός

Ο σκοπός της switch είναι ο έλεγχος της ισότητας μιας έκφρασης σε σχέση με διάφορες σταθερές τιμές, οι οποίες δηλώνονται με τις εντολές case. Όταν βρεθεί η case που ταιριάζει, εκτελούνται οι εντολές από εκεί και κάτω, μέχρι είτε να φτάσει η εκτέλεση στο άγκιστρο του switch που κλείνει, είτε να συναντηθεί η εντολή break (εφόσον υπάρχει). Εάν δεν ταιριάζει η τιμή καμίας case, τότε εκτελούνται οι εντολές που ακολουθούν την default (εφόσον αυτή υπάρχει). Η θέση της default σε σχέση με τις case είναι ελεύθερη.

Σύνταξη

Η σύνταξη της εντολής switch στη C είναι ενδεικτικά:

```
switch (expression) {
    case constant1:
        // εντολές 1
    case constant2:
        // εντολές 2
        break;
    default:
        // εναλλακτικές εντολές – συνήθως η default μπαίνει στο τέλος
        break;
    case constant3:
        // εντολές 2
        break;
}
```

Παραδείγματα

Απλό «calculator»

```
void simpleCalc(char operator, double n1, double n2) {
    double result = 0;
    switch(operator) {
        case '-':
            n2 = -n2;
        case '+':
            result = n1 + n2;
            break;
        case '*':
            result = n1 * n2;
            break;
        case '/':
            if (n2 != 0) {
                result = n1 / n2;
                break;
            }
        default: // operator doesn't match any case constant +, -, *, /
            result = SOME_VALUE_INDICATING_AN_ERROR;
            break;
    }
    return result;
}
```

Επαναλήψεις

Ο βρόχος **for**

Σκοπός

Η εντολή for έχει σκοπό την επανάληψη ενός block εντολών. Οι εντολές αυτές εκτελούνται βάσει μιας συνθήκης (test expression) η οποία ελέγχεται στην αρχή κάθε επανάληψης. Στο τέλος κάθε επανάληψης ορίζεται και μία εντολή (update) η οποία εκτελείται, πριν την επόμενη αξιολόγηση της συνθήκης. Πριν την πρώτη επανάληψη εκτελείται και η εντολή της αρχικοποίησης (initialization).

Εκτός της συνθήκης, οι εντολές αρχικοποίησης και επανάληψης, είναι προαιρετικές.

Σύνταξη

Η σύνταξη του βρόχου for στη C είναι:

```
for (initializationStatement ; testExpression ; updateStatement) {
    // εντολές που αποτελούν το σώμα του βρόχου
}
```

Παραδείγματα

Εκτύπωση ακεραίων από 1 έως N

```
for (int i=0 ; i<N ; ++i) {
    printf(("%d ",i+1);
}
```

Υπολογισμός του N παραγοντικού

```
for (factorial=1,i=2; i=<N ; ++i) {
    factorial*=i;
}
```

Άθροισμα των στοιχείων ενός πίνακα n στοιχείων

```
int sum=0;
for (i=0; i<n ; ++i) {
    sum+=a[i];
}
```

Εκτύπωση των στοιχείων ενός πίνακα με m γραμμές και n στήλες

```
for (j=0; j<m ; ++j) {
    for (i=0; i<n ; ++i) {
        printf("%d ", a[j][i]);
    }
    printf("\n");
}
```

Ο βρόχος **while**

Σκοπός

Ο σκοπός της while είναι η επανάληψη από καμία, έως άπειρες φορές μίας ομάδας εντολών. Στην αρχή κάθε επανάληψης ελέγχεται η συνθήκη και εφόσον είναι αληθής πραγματοποιείται η επανάληψη, αλλιώς σταματά η διαδικασία και η ροή του προγράμματος συνεχίζει μετά από το block εντολών της while. Εάν από παράληψη του προγραμματιστή οι τιμές που εμπλέκονται στη συνθήκη δεν αλλάζουν μέσα στο σώμα του βρόχου, τότε ο βρόχος μπορεί να εκτελείται συνεχώς κάνοντας αδύνατη τη συνέχιση του προγράμματος.

Σύνταξη

Η σύνταξη του βρόχου while στη C είναι:

```
while (testExpression) {
    // εντολές που αποτελούν το σώμα του βρόχου
    // και εκτελούνται όσο η συνθήκη είναι αληθής
}
```

Παραδείγματα

Υπολογισμός του αθροίσματος της ακολουθίας $1 + 1/3 + 1/3^2 + 1/3^3 + \ldots$

```
double x=1.0;
double sum=0.0;
while(x>0.001) {
    sum+=x;
    x/=3;
}
```

Υπολογισμός του αθροίσματος των στοιχείων πίνακα

```
int sum=0;
int data[]={1,2,3,4,5,6,7,8,9,10};
int count =0;

while(count<10) {
    sum+=data[count];
    count++;
}
```

Ο βρόχος **do...while**

Σκοπός

Λειτουργεί όπως ο βρόχος while, όμως οι εντολές εκτελούνται τουλάχιστον μία φορά και στο τέλος της κάθε επανάληψης γίνεται ο έλεγχος της συνθήκης. Όσο η συνθήκη είναι αληθής η διαδικασία επαναλαμβάνεται. Όταν γίνει ψευδής, τότε τερματίζεται η διαδικασία και η εκτέλεση συνεχίζεται μετά την εντολή while.

Σύνταξη

Η σύνταξη του βρόχου do…while στη C είναι:

```
do {
    // εντολές που αποτελούν το σώμα του βρόχου
} while (testExpression) ;
```

Παραδείγματα

Άθροισμα των θετικών αριθμών που εισάγονται από το πληκτρολόγιο

```
int x=0, sum=0;
do {
    sum+=x;
    scanf("%d",&x);
} while(x>0) ;
```

Προσθήκη ενός στοιχείου σε ένα πίνακα εφόσον δεν υπάρχει ήδη

```
do {
    x=get_next_x();
    i=find_in_array(x ,a ,N);
    if(i<0) {
        append_to_array(x ,a ,N);
        N++;
    }
} while( have_more_x() ) ;
```

Εύρεση πρώτου διαθέσιμου ονόματος αρχείου

```
do {
    name=generate_next_name();
} while( file_exist_in_folder(name ,folder) ) ;
```

Εκτύπωση των στοιχείων ενός πίνακα με m γραμμές και n στήλες

```
for (int j=0; j<m ; ++j) {
    for (int i=0; i<n ; ++i)
        printf("%d " , a[i][j]);
    printf("\n");
}
```

Η εντολή **break**

Σκοπός

Ο σκοπός της εντολής `break` είναι να διακόπτει την εκτέλεση ενός βρόχου (ή της εντολής `switch`), μέσα στον οποίο βρίσκεται, και να συνεχίζει η εκτέλεση μετά το τέλος του block του βρόχου. Εάν υπάρχει μία εντολή `break` σε εμφωλευμένους βρόχους, τότε η `break` σχετίζεται με τον πιο εσωτερικό από αυτούς. Επίσης λόγω του τερματισμού εκτέλεσης του block η `break` πρέπει να χρησιμοποιείται πάντα στα πλαίσια κάποιας `if`, αλλιώς οι εντολές του βρόχου από την `break` και κάτω δε θα εκτελούταν ποτέ.

Στην περίπτωση της switch τον ρόλο της `if` τον έχουν οι εντολές `case`.

Σύνταξη

Η σύνταξη της εντολής break στη C είναι:

```
    break;
```

Παραδείγματα

Εκτύπωση περιοχών μονοδιάστατου πίνακα

```
    for (int i=0 ; i<n ; ++i) {
        printf("i: %i ",i);
        for (int j=0 ; j<n ; ++j) {
            if (A[i] > A[j]) {
                break;
            }
            printf("%d ",A[j]);
        }
        // Μετά την break η εκτέλεση συνεχίζεται εδώ
        printf("\n");
    }
```

για πίνακα A = { 1, 2, 3, 4, 3, 2, 3, 4, 3, 2, 1 } το αποτέλεσμα θα είναι:

```
i = 0 : 1 2 3 4 3 2 3 4 3 2 1
i = 1 :
i = 2 :
i = 3 :
i = 4 :
i = 5 :
i = 6 :
i = 7 :
i = 8 :
i = 9 :
i = 10 : 1 2 3 4 3 2 3 4 3 2 1
```

Η εντολή **continue**

Σκοπός

Ο σκοπός της εντολής `continue` είναι να διακόπτει την εκτέλεση της τρέχουσας επανάληψης ενός βρόχου μέσα στον οποίο βρίσκεται και να ξεκινά την επόμενη εφόσον ισχύουν οι προϋποθέσεις κατά τα γνωστά (να είναι αληθής δηλαδή η συνθήκη). Εάν υπάρχει μία εντολή `continue` σε εμφωλευμένους βρόχους, τότε η `continue` σχετίζεται με τον πιο εσωτερικό από αυτούς. Επίσης λόγω του τερματισμού της επανάληψης, η `continue` πρέπει να χρησιμοποιείται πάντα στα πλαίσια κάποιας `if`, αλλιώς οι εντολές όλων των επαναλήψεων από την `continue` και κάτω δε θα εκτελούταν ποτέ.

Σύνταξη

Η σύνταξη της εντολής continue στη C είναι:

```
    continue;
```

Παραδείγματα

Εκτύπωση περιοχών μονοδιάστατου πίνακα

```
    for (int i=0 ; i<n ; ++i) {
        printf("i: %i ",i);
        for (int j=0 ; j<n ; ++j) {
            if (A[i] > A[j]) {
                continue;
            }
            printf("%d ",A[j]);
        }
        // Μετά την break η εκτέλεση συνεχίζεται εδώ
        printf("\n");
    }
```

για πίνακα A = { 1, 2, 3, 4, 3, 2, 3, 4, 3, 2, 1 } το αποτέλεσμα θα είναι:

```
i = 0 : 1 2 3 4 3 2 3 4 3 2 1
i = 1 : 2 3 4 3 2 3 4 3 2
i = 2 : 3 4 3 3 4 3
i = 3 : 4 4
i = 4 : 3 4 3 3 4 3
i = 5 : 2 3 4 3 2 3 4 3 2
i = 6 : 3 4 3 3 4 3
i = 7 : 4 4
i = 8 : 3 4 3 3 4 3
i = 9 : 2 3 4 3 2 3 4 3 2
i = 10 : 1 2 3 4 3 2 3 4 3 2 1
```

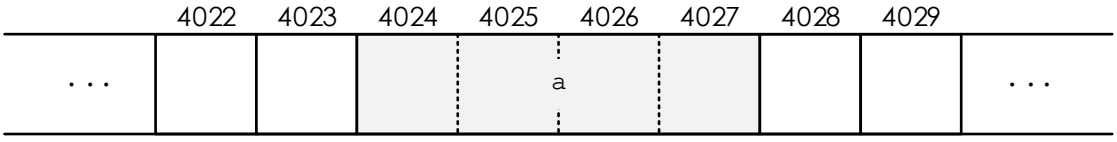
Μνήμη και δείκτες

Δείκτες

Σχηματικά, η μνήμη του υπολογιστή απεικονίζεται (μοντελοποιείται) ως μία ακολουθία από κουτάκια όπου κάθε ένα από αυτά χωράει 1byte (=8bits). Αυτά τα κουτάκια ονομάζονται **θέσεις μνήμης**. Είναι αριθμημένα σε αύξουσα σειρά, ξεκινώντας από το 0. Ο αριθμός που αντιστοιχεί στην κάθε θέση μνήμης ονομάζεται **διεύθυνση της μνήμης**.

Έχει ήδη αναφερθεί ότι οι μεταβλητές οποιουδήποτε απλού ή σύνθετου τύπου δεδομένων, αποθηκεύονται στη μνήμη του υπολογιστή. Η αποθήκευση κάθε εγγενούς μεταβλητής (αλλά και κάθε μεταβλητής σύνθετου τύπου δεδομένων) γίνεται πάντοτε σε διαδοχικές θέσεις μνήμης.

Έτσι για παράδειγμα μία μεταβλητή `a` τύπου `int` χρειάζεται 4 bytes άρα και θέσεις μνήμης για να αποθηκευτεί. Το σύστημα θα επιλέξει κάποιο σημείο της μνήμης για να την αποθηκεύσει. Και το αποτέλεσμα θα είναι π.χ.:



Εδώ φαίνεται ότι η μεταβλητή a έχει αποθηκευτεί στις θέσεις μνήμης από 4024 μέχρι και 4027. Λέμε ότι η μεταβλητή a είναι αποθηκευμένη στην 4024, δηλαδή τη διεύθυνση της πρώτης θέσης μνήμης που χρησιμοποιείται. Το πόσες ακόμα θέσεις μνήμης καταλαμβάνει προκύπτει από τον τύπο δεδομένων της.

Η διεύθυνση της μνήμης στην οποία αποθηκεύεται κάποια μεταβλητή ονομάζεται **δείκτης** (**pointer**) στη μεταβλητή αυτή. Για να βρεθεί αυτή η θέση μνήμης γράφεται:

```
&a
```

Χρησιμοποιείται ο τελεστής **&** που διαβάζεται και address of. Η τιμή αυτή μπορεί να αποθηκευτεί σε κάποια μεταβλητή τύπου δείκτη, της οποίας όμως ο τύπος δεδομένων θα πρέπει να είναι "δείκτης σε ακέραιο" (αφού εδώ η `a` είναι τύπου `int`). Αυτό δηλώνεται όπως και μία ακέραια μεταβλητή, αλλά πριν το όνομα της μεταβλητής γράφεται ένας αστερίσκος.

```
int *Ap;
```

Άρα για να αποθηκευτεί σε αυτή ο δείκτης της μεταβλητής `a` γράφεται:

```
int a;
int *Ap = &a;
```

Ο αρχικός τύπος δεδομένων (εδώ το `int`) σε έναν δείκτη/pointer είναι απαραίτητος ώστε να γνωρίζει η γλώσσα (ο compiler), τι τύπος δεδομένων θα βρεθεί σε εκείνη τη θέση μνήμης που δείχνει ο δείκτης.

Με δεδομένο έναν δείκτη προς μία θέση μνήμης, το περιεχόμενό της (δηλαδή το περιεχόμενο της αρχικής μεταβλητής) γίνεται διαθέσιμο ως:

```
*Ap = 100;
```

Δηλαδή το * (ως μοναδιαίος/unary τελεστής) λειτουργεί αντίστροφα από τον &. Η παραπάνω εντολή δεν αλλάζει το που δείχνει ο pointer, αλλά αλλάζει το περιεχόμενο της μνήμης εκεί που δείχνει ο pointer, δηλαδή τη μεταβλητή a. Βάσει δηλαδή των προηγούμενων παραδειγμάτων, η μεταβλητή a έχει πλέον την τιμή 100.

Προσοχή! Σαν μνημονικό κανόνα μπορεί να θεωρηθεί ότι η δήλωση `int *Ap` μπορεί να διαβαστεί με δύο τρόπους

`int *` Ap ή int `* Ap`

δηλαδή είτε ότι η μεταβλητή Ap είναι pointer σε int, είτε ότι η μεταβλητή *Ap είναι int.

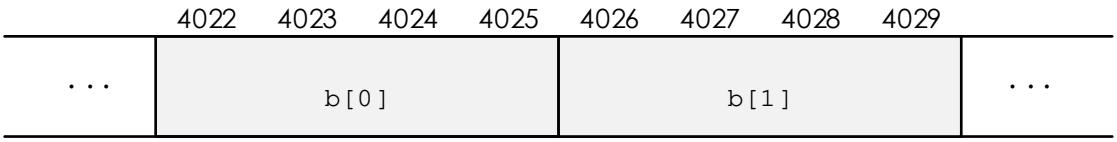
Και σε κάθε περίπτωση μπορώ να έχω είτε την τιμή, είτε τον δείκτη.

Με δεδομένο το..	Γράφω για να βρω..	
	..την τιμή	..τον δείκτη
a	**a**	**&a**
Ap	***Ap**	**Ap**

Για τους σύνθετους τύπους δεδομένων, ακολουθούν οι δύο παρακάτω περιπτώσεις.

Πίνακες

Για τους πίνακες τα στοιχεία είναι συνεχόμενα στη μνήμη. Π.χ. `int b[2];` Άρα:



όπου έχουμε τα δύο στοιχεία διαδοχικά. Προσέξτε τα εξής:

- Αφού το κάθε στοιχείο είναι και μία μεταβλητή μπορούμε να γράψουμε: `&b[0]` ή `&b[1]`
- Οι τιμές των δύο δεικτών διαφέρουν όσο χώρο καταλαμβάνει ένα στοιχείο. Και εδώ που το κάθε στοιχείο είναι int η διαφορά θα είναι `sizeof(int)`
- Ως δείκτη στον πίνακα ονομάζουμε τον δείκτη στο πρώτο στοιχείο του. Δηλαδή το `&b[0]`
- Το ίδιο το όνομα του πίνακα είναι δείκτης στον πίνακα άρα αριθμητικά ταυτίζονται τα b και `&b[0]`

Προσοχή! Το όνομα του πίνακα λειτουργεί και ως δείκτης. Ισχύει και το αντίστροφο. Με δεδομένο έναν δείκτη, αυτός μπορεί να χρησιμοποιηθεί ως πίνακας στη σύνταξη με αγκύλες. Είναι ευθύνη του προγραμματιστή όμως, να υπάρχει όντως πίνακας εκεί που δείχνει ο δείκτης.

Άρα εάν για μία δήλωση

```
int *Bp = &b[0];
```

τότε η τιμή του `Bp` ταυτίζεται με αυτή του b και μπορεί ισοδύναμα να γραφεί π.χ. για το πρώτο στοιχείο του πίνακα:

```
Bp[0]
```

καθώς και η παρακάτω έκφραση είναι αληθής για οποιοδήποτε i:

```
Bp[i] == b[i]
```

Δομές

Οι δομές δεδομένων αποθηκεύονται αντίστοιχα. Π.χ. Για μια δομή όπως η παρακάτω:

```
typedef struct _record {
  char type;
  char series;
  short num;
  char label[10];
} Record;

Record r;
Record Rp = &r;
```

Εάν μια τέτοια δομή `r` είναι αποθηκευμένη σε κάποιο σημείο της μνήμης τότε θα είναι αποθηκευμένη έτσι:

Προσέξτε ότι τα πεδία της δομής είναι αποθηκευμένα διαδοχικά όπως και στη δήλωσή της.

Η αναφορά στα πεδία/μέλη της δομής όταν διαθέσιμος είναι ένας pointer θα έπρεπε να γραφεί ως εξής:

```
( *Rp ).type
```

Επειδή η . έχει μεγαλύτερη προτεραιότητα από το *, χρειάζονται και οι παρενθέσεις. Το * μετατρέπει τον δείκτη σε δομή και μετά η τελεία χρησιμοποιείται για να γίνει η αναφορά στο πεδίο.

Για να αποφευχθεί αυτή η γραφή, η C παρέχει έναν άλλο τελεστή που γράφεται με δύο χαρακτήρες. Τον –> ο οποίος είναι ισοδύναμος με την προηγούμενη γραφή, και με τον οποίο γράφεται πιο ξεκάθαρα:

```
Rp->type
```

Άρα στην παραπάνω έκφραση το `Rp` θα πρέπει να είναι δείκτης σε δομή και το `type` το όνομα του πεδίου.

Δείκτες δεικτών

Οι δείκτες όπως είδαμε είναι μεταβλητές που μέσα τους αποθηκεύουν τη διεύθυνση της μνήμης μιας άλλης μεταβλητής. Άρα μπορούμε με την ίδια λογική να έχουμε δείκτες σε δείκτες.

```
int a = 50;
int *Ap = &a;
int **App = &Ap;
**App = 100; // Αλλάζει η τιμή της a από 50 σε 100
```

και ακόμα με ακόμα μεγαλύτερη πολλαπλότητα (δείκτη σε δείκτη σε δείκτη…). Επίσης λόγω του δυισμού πίνακα και δείκτη, οι δείκτες σε δείκτες (...) λειτουργούν όπως οι πολυδιάστατοι πίνακες.

Διαχείριση μνήμης

Εκτός από την απλή μέθοδο δήλωσης μεταβλητών που εξασφαλίζει την απαραίτητη μνήμη γι' αυτές, υπάρχει και ένας άλλος τρόπος να δηλωθούν. Η μέθοδος αυτή έχει νόημα και εφαρμόζεται για μεγαλύτερα κομμάτια μνήμης και όχι για απλές native/εγγενείς μεταβλητές. Εφαρμόζεται σε σύνθετους τύπους δεδομένων.

Για να αποθηκευτεί στη μνήμη ένας πίνακας από `N` στοιχεία του τύπου `Record` (του προηγούμενου παραδείγματος) θα χρειαστούν `N` φορές το μέγεθος ενός `Record`. Για να βρεθεί αυτό το μέγεθος χρησιμοποιείται ο τελεστής `sizeof` ο οποίος επιστρέφει το μέγεθος σε bytes που απαιτείται για την αποθήκευση ενός τύπου δεδομένων.

Η εντολή που μπορεί να παρέχει μνήμη είναι η `malloc` η οποία δέχεται μία παράμετρο (το πλήθος της μνήμης σε bytes) και επιστρέφει έναν δείκτη (ως `void *`) σε αυτή την περιοχή. Εάν δεν γίνεται να βρεθεί κατάλληλη μνήμη τόσο μεγάλη όσο ζητήθηκε, τότε επιστρέφεται μηδενική τιμή. Άρα για το παραπάνω παράδειγμα:

```
Record *records = (Record *)malloc( sizeof(Record) * N );
```

Προσοχή! Αμέσως μετά από κάθε κλήση της `malloc` θα πρέπει να γίνεται έλεγχος εάν το αποτέλεσμα ήταν μηδενικό. Σε μια τέτοια περίπτωση δεν πρέπει να χρησιμοποιηθεί το αποτέλεσμα της κλήσης, αλλά ο αλγόριθμος να λάβει υπόψη του ότι τελείωσε η διαθέσιμη μνήμη.

Προσοχή! Για να είναι πιο σωστή η δήλωση και να μην προκύπτουν ενδεχόμενα μηνύματα από τον compiler (μεταφραστή), θα πρέπει το αποτέλεσμα της malloc να γίνεται cast (όπως παραπάνω) στον σωστό τύπο δεδομένων.

Αφού ολοκληρωθεί η χρήση της περιοχής μνήμης που ζητήθηκε θα πρέπει **οπωσδήποτε**, να απελευθερωθεί αυτή η μνήμη με τη χρήση της εντολής `free`. Μετά την κλήση της `free` οι δείκτες/pointers που έδειχναν σε αυτή τη μνήμη δεν πρέπει να

χρησιμοποιηθούν καθώς πιθανότατα θα οδηγήσουν στην διακοπή της εκτέλεσης του προγράμματος.

```
free(records);
// από εδώ και κάτω δεν πρέπει να χρησιμοποιηθεί πλέον η records
```

Προσοχή! Εάν για οποιοδήποτε λόγο δεν απελευθερωθεί η μνήμη αυτή, τότε δεν θα είναι διαθέσιμη στον Η/Υ μέχρι τον τερματισμό του προγράμματος. Ενδεχομένως να μην είναι διαθέσιμη και μέχρι τον τερματισμό του λειτουργικού συστήματος (reboot).

Κλήση με αναφορά/Call by reference

Μία άμεση συνέπεια της χρήσης των δεικτών είναι ότι μπορεί να παρακαμφθεί όπου είναι επιθυμητό, και μάλιστα με ρητό τρόπο, ο περιορισμός της κλήσης με τιμή (call by value). Για παράδειγμα, μια συνάρτηση swap η οποία θα ανταλλάσσει τα περιεχόμενα δύο μεταβλητών που θα της δίνονται, είναι πλέον εφικτό να γραφεί, ως ακολούθως:

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

και η κλήση της θα γίνεται ως εξής:

```
int x = 10, y = 20;
swap(&x, &y);
```

Μετά την εκτέλεση του παραπάνω κώδικα οι τιμές των `x` και `y` θα έχουν ανταλλαχθεί μεταξύ τους.

ΣΗΜΕΙΩΣΕΙΣ

Προεπεξεργαστής

Ο προεπεξεργαστής είναι το πρώτο βήμα στη διαδικασία της μετατροπής του κώδικα από μορφή κειμένου, σε εκτελέσιμο αρχείο. Η διαδικασία ονομάζεται building και έχει τα εξής στάδια:

1. Preprocessing = Μετασχηματισμός του κειμένου του κώδικα
2. Compilation = Μετατροπή του κειμένου του κώδικα σε assembly
3. Assembling = Μετατροπή της assembly σε γλώσσα μηχανής (object code)
4. Linking = Σύνδεση του κώδικα (object code) με τις βιβλιοθήκες

Ο σκοπός του είναι να μεταγράψει τον κώδικα πάλι σε μορφή κειμένου, δίνοντας δυνατότητες στον προγραμματιστή να ελέγξει τη μετατροπή αυτή. Οι εντολές του προεπεξεργαστή ξεχωρίζουν επειδή ξεκινούν με το σύμβολο της δίεσης #. Ονομάζονται ψευδοεντολές καθώς δεν παράγουν εκτελέσιμο κώδικα καθαυτές, αλλά μόνο καθοδηγούν τον προεπεξεργαστή στο πως θα μετασχηματίσει το κείμενο του κώδικα. Δηλαδή το αποτέλεσμα της εκτέλεσης του προεπεξεργαστή είναι και πάλι κώδικας σε μορφή κειμένου.

Οι συνηθέστερα χρησιμοποιούμενες εντολές του preprocessor είναι οι:

`#include` , η οποία συντάσσεται ως εξής:

```
#include <somefile.h>        ή        #include "somefile.h"
```

Αυτή μεταφέρει το περιεχόμενο του αρχείου somefile.h στη θέση της ψευδοεντολής αυτής. Η διαφορά ανάμεσα στις δύο γραφές είναι ότι όταν χρησιμοποιούνται τα < και > για να οριοθετήσουν το όνομα του αρχείου, η αναζήτηση γίνεται σε συγκεκριμένους φακέλους από τον compiler. Ενώ όταν χρησιμοποιούνται τα εισαγωγικά " η αναζήτηση περιλαμβάνει επιπλέον και τον φάκελο που περιέχει τον κώδικα.

`#define` , η οποία συντάσσεται ως εξής:

```
#define IDENTIFIER expression                    ή
#define IDENTIFIER(x,..) expression_of_x..
```

Αυτή η ψευδοεντολή αντικαθιστά το `IDENTIFIER` σε όλη την έκταση του κώδικα με την `expression`, ακριβώς όπως λειτουργεί και η εύρεση/αντικατάσταση (find/replace) σε έναν επεξεργαστή κειμένου.

Εάν συνεχόμενα (χωρίς κανένα χαρακτήρα) στο `IDENTIFIER` δοθούν μέσα σε παρενθέσεις παράμετροι, τότε προκύπτει ένα «συναρτησιοειδές» το οποίο εκτός από την αντικατάσταση της απλής περίπτωσης αντικαθιστά τις τιμές των ορισμάτων μέσα στο `expression`. Επειδή η αντικατάσταση γίνεται «τυφλά» θα πρέπει μέσα στην παράσταση `expression`, οι παράμετροι να εμφανίζονται πάντα σε παρενθέσεις. Αυτό είναι καλή πρακτική καθώς δεν είναι περιορισμός λόγω της ψευδοεντολής, αλλά επειδή συνήθως είναι αυτό που θέλει ο προγραμματιστής.

Εάν μέσα στην έκφραση δύο παράμετροι πρέπει να ενωθούν χωρίς μεταξύ τους κάποιο χαρακτήρα τότε μπορεί να γραφεί ανάμεσά τους ο τελεστής ##

`#if` , **`#ifdef`** , **`#ifndef`** , **`#else`** , **`#elif`** , **`#endif`** οι οποίες επιτρέπουν την επιλεκτική συμπερίληψη τμημάτων κώδικα στο compilation.

Τυπικές βιβλιοθήκες της C (C standard libraries)

Μαθηματικά <math.h>

Η βιβλιοθήκη αυτή παρέχει μαθηματικές σταθερές και μαθηματικές συναρτήσεις για διάφορες λειτουργίες. Οι βασικότερες από αυτές παρουσιάζονται ακολούθως ομαδοποιημένες βάσει της λειτουργίας τους.

Σταθερές	
M_E	Η τιμή του **e** (2.718...)
M_PI	Η τιμή του **π** (3.14159...)
M_SQRT2	Η τιμή του $\sqrt{2}$ (1.414...)
M_LN2	Η τιμή του $\ln 2$ (0.6931...)
Γενικά Μαθηματικά	
`double` **`fabs`**`(double x)`	Επιστρέφει την **απόλυτη τιμή** του x
`double` **`sqrt`**`(double x)`	Επιστρέφει την **τετραγωνική ρίζα** του x
`double` **`pow`**`(double x, double y)`	Επιστρέφει το **x υψωμένο στο y**
`double` **`log`**`(double x)`	Ο φυσικός (Νεπέριος) **λογάριθμος** του x
`double` **`log10`**`(double x)`	Ο λογάριθμος του x ως **προς το 10**.
`double` **`exp`**`(double x)`	Ο αριθμός **e υψωμένος στην x**
`double` **`fmod`**`(double x, double y)`	Το **υπόλοιπο της διαίρεσης** x δια y
Στρογγύλευση	
`double` **`ceil`**`(double x)`	**Στρογγυλοποιεί** την τιμή του x στο μικρότερο ακέραιο που είναι **μεγαλύτερος ή ίσος με το x**
`double` **`floor`**`(double x)`	**Στρογγυλοποιεί** την τιμή του x στο μεγαλύτερο ακέραιο που είναι **μικρότερος ή ίσος με το x**

Τριγωνομετρικές συναρτήσεις	
`double` **`sin`**`(double x)`	Το **ημίτονο** της x, όταν η χ είναι εκφρασμένη σε ακτίνια (rad)
`double` **`cos`**`(double x)`	Το **συνημίτονο** της x, όταν η χ είναι εκφρασμένη σε ακτίνια (rad)
`double` **`tan`**`(double x)`	Η **εφαπτομένη** της x, όταν η χ είναι εκφρασμένη σε ακτίνια (rad)
Υπερβολικές τριγωνομετρικές συναρτήσεις	
`double` **`sinh`**`(double x)`	Το **υπερβολικό ημίτονο** της x, όταν η χ είναι εκφρασμένη σε ακτίνια (rad)
`double` **`cosh`**`(double x)`	Το **υπερβολικό συνημίτονο** της x, όταν η χ είναι εκφρασμένη σε ακτίνια (rad)
`double` **`tanh`**`(double x)`	Η **υπερβολική εφαπτομένη** της x, όταν η χ είναι εκφρασμένη σε ακτίνια (rad)
Αντίστροφες τριγωνομετρικές συναρτήσεις	
`double` **`asin`**`(double x)`	Το **τόξο ημιτόνου** του x, εκφρασμένο σε ακτίνια (rad) στο διάστημα [-π/2 , π/2]
`double` **`acos`**`(double x)`	Το **τόξο συνημιτόνου** του x, εκφρασμένο σε ακτίνια (rad) στο διάστημα [0, π]
`double` **`atan`**`(double x)`	Η **τόξο εφαπτομένης** του x, εκφρασμένη σε ακτίνια (rad) στο διάστημα [-π/2 , π/2]
`double` **`atan2`**`(double y, double x)`	Η **τόξο εφαπτομένης** που προκύπτει από τη διαίρεση y/x, **λαμβάνοντας υπόψη και τα πρόσημα** ώστε να υπολογιστεί η γωνία στο σωστό τεταρτημόριο. Επιστρέφεται μία τιμή στο διάστημα [0, 2π)

Συναρτήσεις εισόδου/εξόδου <stdio.h>

Συναρτήσεις για την σύνδεση με τις μονάδες εισόδου/εξόδου (πληκτρολόγιο, οθόνη, αρχεία, κλπ).

Σταθερές και συναρτήσεις που ορίζονται είναι:

Σταθερές & τύποι δεδομένων	
FILE	Ο τύπος δεδομένων περιγράφει ένα ανοιχτό αρχείο.
EOF	Η σταθερά αυτή είναι μία ειδική τιμή που παριστάνει το τέλος ενός αρχείου.
SEEK_CUR, **SEEK_END** και **SEEK_SET**	Σταθερές που χρησιμοποιούνται στη συνάρτηση `fseek`.
stdin	Σταθερά που παριστάνει την τυπική είσοδο, συνήθως το πληκτρολόγιο
stdout	Σταθερά που παριστάνει την τυπική έξοδο, συνήθως την οθόνη και αφορά τα μηνύματα κανονικής λειτουργίας ενός λογισμικού
stderr	Σταθερά που παριστάνει την τυπική έξοδο σφαλμάτων, συνήθως την οθόνη, και αφορά τα μηνύματα σφαλμάτων ενός λογισμικού
Άνοιγμα / Κλείσιμο αρχείων	
`FILE *fopen( const char *filename, const char *mode)`	Δοκιμάζει να ανοίξει το αρχείο με όνομα ή διαδρομή (path) `filename` για λειτουργία βάσει της παραμέτρου `mode`. Επιστρέφει τον περιγραφέα αρχείου (file descriptor) ή `NULL` εάν αποτύχει το άνοιγμα.
`int fclose(FILE *stream)`	Κλείνει το αρχείο stream, το οποίο προηγουμένως έχει ανοιχτεί με την `fopen`. Πριν κλείσει καλείται "κρυφά" η `fflush`.
`char *tmpnam(char *str)`	Επιστρέφει ένα πρόχειρο όνομα αρχείου που δεν υπάρχει ήδη, ως προς τον πρόχειρο φάκελο του συστήματος. Η τιμή επιστρέφεται από τη συνάρτηση. Εάν η παράμετρος δεν είναι `NULL` αντιγράφεται και εκεί που υποδεικνύει η παράμετρος.

Συναρτήσεις μορφοποιημένης εξόδου	
`int printf( const char *format, ... )`	Η συνάρτηση αυτή εκτυπώνει μήνυμα στην τυπική έξοδο (`stdout`). Στο μήνυμα επιτρέπεται να περιέχονται και ειδικές ακολουθίες χαρακτήρων οι οποίοι υποδεικνύουν τα σημεία στα οποία θα τοποθετηθούν οι τιμές από τα επόμενα ορίσματα. Οι ακολουθίες αυτές ξεκινούν πάντα με τον χαρακτήρα `%`. Δείτε το σχετικό παράρτημα. Για κάθε τέτοια ακολουθία θα πρέπει να αντιστοιχεί και μία ακόμα μεταβλητή που ακολουθεί το 1ο όρισμα.
`int sprintf(char *str, const char *format, ... )`	Η συνάρτηση αυτή λειτουργεί όπως η `printf` με ένα επιπλέον όρισμα στην αρχή. Το όρισμα αυτό είναι μία συμβολοσειρά (η οποία θα πρέπει να έχει επαρκή χώρο). Το κείμενο που θα εμφανιζόταν στην οθόνη τοποθετείται μέσα στη συμβολοσειρά αυτή.
`int fprintf( FILE *stream, const char *format, ... )`	Η συνάρτηση αυτή λειτουργεί όπως η `printf` με ένα επιπλέον όρισμα στην αρχή. Το όρισμα αυτό είναι τύπου δεδομένων `FILE`. Το κείμενο που θα εμφανιζόταν στην οθόνη εγγράφεται στην "τρέχουσα θέση" μέσα σε αυτό το αρχείο.
Συναρτήσεις μορφοποιημένης εισόδου	
`int scanf( const char *format, ... )`	Η συνάρτηση αυτή διαβάζει από την τυπική είσοδο (`stdin`). Το διάβασμα αυτό υπακούει σε κανόνες μορφοποίησης παρόμοιους με αυτούς τους `printf`. Δείτε το αντίστοιχο παράρτημα. Για κάθε ακολουθία χαρακτήρων μορφοποίησης θα πρέπει να δίνεται ένας δείκτης σε κατάλληλου τύπου μεταβλητή.
`int sscanf( const char *str, const char *format, ... )`	Η συνάρτηση αυτή λειτουργεί όπως η `scanf` με ένα επιπλέον όρισμα στην αρχή. Το όρισμα αυτό είναι μία συμβολοσειρά. Από τη συμβολοσειρά αυτή, αντί του πληκτρολογίου, προέρχονται οι τιμές που θα τοποθετηθούν στις μεταβλητές.
`int fscanf( FILE *stream, const char *format, ... )`	Η συνάρτηση αυτή λειτουργεί όπως η `scanf` με ένα επιπλέον όρισμα στην αρχή. Το όρισμα αυτό είναι ένα "αρχείο". Από το αρχείο αυτό, αντί του πληκτρολογίου, προέρχονται οι τιμές που θα τοποθετηθούν στις μεταβλητές.

Συναρτήσεις μη μορφοποιημένης εισόδου και εξόδου	
`char *fgets(char *str, int n, FILE *stream)`	Διαβάζει μία γραμμή με το πολύ n-1 χαρακτήρες, από το αρχείο `stream` μέσα στο κείμενο `str`. Εάν διαβαστεί έστω και ένας χαρακτήρας επιστρέφει το `str` και ως αποτέλεσμα, αλλιώς επιστρέφει `NULL`.
`int getc(FILE *stream)`	Διαβάζει έναν χαρακτήρα από το αρχείο stream
`int ungetc(int char, FILE *stream)`	"Επιστρέφει" έναν χαρακτήρα που διαβάστηκε προηγουμένως στο αρχείο `stream`. Προσοχή! Δεν γράφει στο αρχείο τον χαρακτήρα.
`int fputs( const char *str, FILE *stream)`	Γράφει το κείμενο `str` στο αρχείο `stream`, χωρίς τον μηδενικό τελικό χαρακτήρα. Εάν όλα πάνε καλά επιστρέφει θετική τιμή.
`int putc(int char, FILE *stream)`	Γράφει τον χαρακτήρα `char` στο αρχείο που περιγράφει η `stream`.
Δυαδική είσοδος/έξοδος	
`size_t fread( void *ptr, size_t size, size_t nmemb, FILE *stream)`	Διαβάζει από το αρχείο stream ως δυαδικά δεδομένα, κομμάτια μεγέθους `nmemb` και σε ποσότητα μέχρι και `size`. Τα τοποθετεί στη θέση μνήμης `ptr`. Επιστρέφει το πλήθος των κομματιών που διαβάστηκαν επιτυχώς.
`size_t fwrite( const void *ptr, size_t size, size_t nmemb, FILE *stream)`	Γράφει στο αρχείο stream, από τη θέση μνήμης `ptr`, ως δυαδικά δεδομένα, μέχρι `nmemb` κομμάτια μεγέθους size. Επιστρέφει το πλήθος των κομματιών που γράφτηκαν επιτυχώς.
Εσωτερική κατάσταση αρχείων	
`int fflush(FILE *stream)`	Αναγκάζει την προσωρινή μνήμη (buffer) του αρχείου stream να γραφτεί άμεσα στον δίσκο.
`long int ftell( FILE *stream)`	Επιστρέφει την τρέχουσα θέση του δρομέα στο αρχείο stream μετρώντας από την αρχή του αρχείου.
`int fseek( FILE *stream, long int offset, int whence)`	Μετακινεί το δρομέα του αρχείου stream σε σχέση με το τέλος (`SEEK_END`), την αρχή (`SEEK_SET`) ή την τρέχουσα θέση του (`SEEK_CUR`).
`int feof(FILE *stream)`	Επιστρέφει αληθές εάν ο δρομέας του αρχείου έχει φτάσει στο τέλος του, αλλιώς επιστρέφει ψευδές.
`void rewind(FILE *stream)`	Μεταφέρει τον δρομέα στην αρχή του αρχείου.

Κείμενα <string.h>

Παρέχει συναρτήσεις που αφορούν κείμενα.

Διάφορες συναρτήσεις	
`size_t` **`strlen`**`(const char *str)`	Επιστρέφει το μήκος της συμβολοσειράς που δίνεται ως παράμετρος.
Αναζήτηση σε κείμενο	
`char *`**`strchr`**`(const char *str, int c)`	Βρίσκει τον χαρακτήρα c στο κείμενο `str` ή επιστρέφει `NULL` εάν δεν βρεθεί
`char *`**`strstr`**`(const char *haystack, const char *needle)`	Βρίσκει την πρώτη εμφάνιση του δεύτερου κειμένου μέσα στο πρώτο και επιστρέφει τον δείκτη σε εκείνο το σημείο ή NULL εάν δεν υπάρχει.
`char *`**`strpbrk`**`(const char *str1, const char *str2)`	Βρίσκει τον πρώτο χαρακτήρα στο πρώτο κείμενο (`str1`) από αυτούς που υπάρχουν στο δεύτερο κείμενο και επιστρέφει τον δείκτη σε εκείνο το σημείο ή `NULL` αν δεν υπάρχει κανένας.
Αντιγραφή κειμένων	
`char *`**`strcpy`**`(char *dest, const char *src)`	Αντιγράφει το 2ο κείμενο πάνω στο 1ο (υποθέτοντας ότι υπάρχει επαρκής χώρος)
`char *`**`strncpy`**`(char *dest, const char *src, size_t n)`	Αντιγράφει το 2ο κείμενο πάνω στο 1ο μέχρι και τους n χαρακτήρες
Σύγκριση κειμένων	
`int` **`strcmp`**`(const char *str1, const char *str2)`	Συγκρίνει τα δύο κείμενα και επιστρέφει 0 εάν είναι ίσα, θετική τιμή εάν το 1ο κείμενο είναι λεξικογραφικά μεγαλύτερο από το 2ο και αρνητική εάν είναι μικρότερο.
`int` **`strncmp`**`(const char *str1, const char *str2, size_t n)`	Λειτουργεί όπως και η `strcmp` αλλά συγκρίνει το πολύ μέχρι n χαρακτήρες των δύο κειμένων.
Ένωση κειμένων	
`char *`**`strcat`**`(char *dest, const char *src)`	Ενώνει δύο κείμενα στη θέση του πρώτου. Δηλαδή μετά το τέλος του 1ου κειμένου αντιγράφει το 2ο κείμενο.
`char *`**`strncat`**`(char *dest, const char *src, size_t n)`	Ενώνει δύο κείμενα στη θέση του πρώτου. Δηλαδή μετά το τέλος του 1ου κειμένου αντιγράφει το 2ο κείμενο. Αντιγράφονται το πολύ μέχρι n χαρακτήρες.

Γενικές συναρτήσεις <stdlib.h>

Περιέχει διάφορες λειτουργίες γενικού ενδιαφέροντος, όπως και σταθερές.

Σταθερές & τύποι δεδομένων	
NULL	Η σταθερά αυτή παριστάνει τη μηδενική τιμή αλλά σε τύπο δείκτη (void *)
RAND_MAX	Η σταθερά αυτή παριστάνει τη μέγιστη τιμή που μπορεί να επιστρέψει η συνάρτηση rand
size_t	Ο τύπος δεδομένων size_t είναι μία μορφή ακεραίου κατάλληλη ώστε να μπορεί να εκφραστεί το μέγεθος της μνήμης εκάστου συστήματος.
Συναρτήσεις μετατροπής	
`int atoi(const char *str)`	Επιστρέφει ως int την τιμή του ακεραίου που βρίσκει στο κείμενο str
`long atol(const char *str)`	Επιστρέφει ως long την τιμή του ακεραίου που βρίσκει στο κείμενο str
`double atof(const char *str)`	Επιστρέφει ως double την τιμή του αριθμού κινητής υποδια-στολής που βρίσκει στο κείμενο str
Συναρτήσεις διαχείρισης μνήμης	
`void *malloc(size_t size)`	Δεσμεύει μνήμη σε μέγεθος size σε bytes (θέσεις μνήμης) και επιστρέφει έναν δείκτη στη δεσμευμένη περιοχή. Εάν δεν υπάρχει αρκετή μνήμη τότε επιστρέφει NULL
`void *realloc(void *ptr, size_t size)`	Αλλάζει το μέγεθος της μνήμης που έχει προηγουμένως δεσμευθεί με την malloc ή άλλη κλήση της realloc. Το σύστημα αποφασίζει εάν το νέο μέγεθος θα είναι στην ίδια θέση ή σε άλλη. Η νέα θέση (ίδια ή διαφορετική) επιστρέφεται από την realloc. Εάν ζητηθεί περισσότερη μνήμη αλλά δεν είναι διαθέσιμη επιστρέφεται η τιμή NULL.
`void free(void *ptr)`	Αποδεσμεύει τη μνήμη που προηγουμένως έχει δεσμευθεί από την malloc ή την realloc.

Διάφορες συναρτήσεις	
`void` **`exit`**`(int status)`	Τερματίζει την εκτέλεση του προγράμματος με κωδικό εξόδου status.
`int` **`abs`**`(int x)`	Επιστρέφει την απόλυτη τιμή του ακεραίου int x
`long int` **`labs`**`(long int x)`	Επιστρέφει την απόλυτη τιμή του ακεραίου long x
`int` **`rand`**`(void)`	Παράγει τον επόμενο ψευδοτυχαίο ακέραιο μεταξύ 0 και RAND_MAX
`void` **`srand`**`(unsigned int seed)`	Αλλάζει την τιμή του seed για τη γεννήτρια ψευδοτυχαίων αριθμών. Συνήθως χρησιμοποιείται σε συνδυασμό με κάποια συνάρτηση που παράγει μια τιμή που δεν μπορεί εύκολα να προβλεφθεί (π.χ. την τρέχουσα χρονική στιγμή).

Τύποι χαρακτήρων <ctype.h>

Οι χαρακτήρες (τουλάχιστον οι ASCII χαρακτήρες με κωδικούς 0-127) διακρίνονται σε διάφορες κατηγορίες ή ομάδες οι οποίες έχουν χρηστικό σκοπό. Για την κάθε ομάδα χαρακτήρων υπάρχει μια συνάρτηση που της δίνεται ένας χαρακτήρας και επιστρέφει αληθές (μη μηδενική τιμή) εάν ανήκει στην ομάδα αυτή, αλλιώς επιστρέφει ψευδές (μηδενική τιμή).

Ψηφία / Digits
Τα αριθμητικά ψηφία `0 1 2 3 4 5 6 7 8 9`
Συνάρτηση: `int` **`isdigit`**`(int c)`

Δεκαεξαδικά ψηφία / Hexadecimal digits
Τα αριθμητικά ψηφία και οι χαρακτήρες a-f
`0 1 2 3 4 5 6 7 8 9 A a B b C c D d E e F f`
Συνάρτηση: `int` **`isxdigit`**`(int c)`

Πεζά γράμματα / Lowercase letters
Οι πεζοί χαρακτήρες του Αγγλικού αλφαβήτου
`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`
Συνάρτηση: `int` **`islower`**`(int c)`

Κεφαλαία γράμματα / Uppercase letters
Οι κεφαλαίοι χαρακτήρες του Αγγλικού αλφαβήτου
`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`
Συνάρτηση: `int` **`isupper`**`(int c)`

Γράμματα ή Αλφαβητικοί χαρακτήρες / Letters ή Alphabetic characters
Τα κεφαλαία και τα πεζά γράμματα μαζί
Συνάρτηση: `int` **`isalpha`**`(int c)`

Αλφαριθμητικοί χαρακτήρες / Alphanumeric characters
Τα γράμματα μαζί με τα (αριθμητικά) ψηφία
Συνάρτηση: `int` **`isalnum`**`(int c)`

Σημεία στίξης / Punctuation characters
Οι χαρακτήρες
`! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~`
Συνάρτηση: `int` **`ispunct`**`(int c)`

Γραφικοί χαρακτήρες / Graphical characters
Οι αλφαριθμητικοί χαρακτήρες μαζί με τα σημεία στίξης. Δηλαδή όλοι οι χαρακτήρες οι οποίοι έχουν οπτική αναπαράσταση (εμφανίζουν κάτι στην οθόνη).
Συνάρτηση: `int` **`isgraph`**`(int c)`

Κενοί χαρακτήρες / Space ή Whitespace characters
Χαρακτήρες που έχουν επίδραση στο αποτέλεσμα αλλά δεν έχουν ορατό περιεχόμενο. Πιο συγκεκριμένα είναι το κενό ( ), το οριζόντιο tab (`\t`), το κατακόρυφο tab (`\v`), αλλαγή γραμμής (`\n`), επιστροφή (`\r`) και form feed (`\f`).
Συνάρτηση: `int` **`isspace`**`(int c)`

Εκτυπώσιμοι χαρακτήρες / Printable characters
Οι αλφαριθμητικοί χαρακτήρες και οι κενοί μαζί.
Συνάρτηση: `int` **`isprint`**`(int c)`

Χαρακτήρες ελέγχου / Control characters
Χαρακτήρες με κωδικό 0 (\000) ως και 31 (\037) και ο 127 (\177)
Συνάρτηση: `int` **`iscntrl`**`(int c)`

Χρόνος <time.h>

Συναρτήσεις σχετικά με τον χρόνο.

Μεταξύ άλλων ορίζει τον τύπο **time_t** ο οποίος είναι ακέραιος κατάλληλος να αποθηκεύσει την τρέχουσα χρονική στιγμή.

Προσδιορισμός χρόνου	
`time_t` **`time`**`(time_t *timer)`	Επιστρέφει την τρέχουσα χρονική στιγμή ως αριθμό δευτερολέπτων που έχουν περάσει από το Epoch, δηλαδή από τα μεσάνυχτα της 1ης Ιανουαρίου 1970. Συνήθως ως παράμετρος δίνεται το NULL. Εάν δοθεί μεταβλητή, τότε η τρέχουσα χρονική στιγμή αποθηκεύεται στην παράμετρο (εδώ timer).

ΣΗΜΕΙΩΣΕΙΣ

Παραρτήματα

Παράρτημα Ι - Πίνακας ASCII

Στο κάθε κελί υπάρχει ο αντίστοιχος χαρακτήρας. Ο κωδικός του στο ASCII, γραμμένος σε δεκαεξαδική μορφή, είναι διψήφιος. Το πρώτο (αριστερό ψηφίο) ορίζεται από την στήλη και το 2ο (δεξιό ψηφίο) από τη γραμμή. Κάτω από τον χαρακτήρα υπάρχει ο κωδικός στο δεκαδικό σύστημα.

Για παράδειγμα ο χαρακτήρας E έχει κωδικό $45_{(16)}$ (ή 0x45) και στο δυαδικό 0100 0101.

Bits 4-6 ➔		000	001	010	011	100	101	110	111
Bits 0-3 ↓		0	1	2	3	4	5	6	7
0000	0	NUL ή '\0' 00	DLE 16	**SPC (Κενό)** 32	**0** 48	**@** 64	**P** 80	**`** 96	**p** 112
0001	1	SOH 01	DC1 17	**!** 33	**1** 49	**A** 65	**Q** 81	**a** 97	**q** 113
0010	2	STX 02	DC2 18	**"** 34	**2** 50	**B** 66	**R** 82	**b** 98	**r** 114
0011	3	ETX 03	DC3 19	**#** 35	**3** 51	**C** 67	**S** 83	**c** 99	**s** 115
0100	4	EOT 04	DC4 20	**$** 36	**4** 52	**D** 68	**T** 84	**d** 100	**t** 116
0101	5	ENQ 05	NAK 21	**%** 37	**5** 53	**E** 69	**U** 85	**e** 101	**u** 117
0110	6	ACK 06	SYN 22	**&** 38	**6** 54	**F** 70	**V** 86	**f** 102	**v** 118
0111	7	BEL 07	ETB 23	**'** 39	**7** 55	**G** 71	**W** 87	**g** 103	**w** 119
1000	8	BS 08	CAN 24	**(** 40	**8** 56	**H** 72	**X** 88	**h** 104	**x** 120
1001	9	HT (Tab) ή '\t' 09	EM 25	**)** 41	**9** 57	**I** 73	**Y** 89	**i** 105	**y** 121
1010	A	LF (Enter) ή '\n' 10	SUB 26	***** 42	**:** 58	**J** 74	**Z** 90	**j** 106	**z** 122
1011	B	VT ή '\v' 11	ESC 27	**+** 43	**;** 59	**K** 75	**[** 91	**k** 107	**{** 123
1100	C	FF ή '\f' 12	FS 28	**,** 44	**<** 60	**L** 76	**** 92	**l** 108	**|** 124
1101	D	CR (Enter) ή '\r' 13	GS 29	**-** 45	**=** 61	**M** 77	**]** 93	**m** 109	**}** 125
1110	E	SO 14	RS 30	**.** 46	**>** 62	**N** 78	**^** 94	**n** 110	**~** 126
1111	F	SI 15	US 31	**/** 47	**?** 63	**O** 79	**_** 95	**o** 111	DEL 127

Παράρτημα ΙΙ - Ενδεικτικά μεγέθη και όρια εγγενών τύπων δεδομένων

Στον πίνακα παρουσιάζονται η μέγιστη και η ελάχιστη τιμή που μπορεί να παρασταθεί με κάθε εγγενή τύπο δεδομένων, προσημασμένο ή απρόσημο, ακέραιο ή δεκαδικό.

Τύπος	Τυπικά bits	**Απρόσημες / unsigned**		
		Min	Max	Format string
Ακέραιες τιμές				
char	8	0	255	%u , %o , %x
int	32	0	4294967295	%u , %o , %x
short	16	0	65535	%u , %o , %x
long	32	0	4294967295	%lu , %lo , %lx
long long	64	0	18446744073709551615	%llu,%llo,%llx
Τύπος	Τυπικά bits	**Προσημασμένες / signed**		
		Min	Max	Format string
Ακέραιες τιμές				
char	8	-128	127	%d , %i
int	32	-2147483648	2147483647	%d , %i
short	16	-32768	32767	%d , %i
long	32	-2147483648	2147483647	%ld
long long	64	-9223372036854775808	9223372036854775807	%lld
Δεκαδικές (κινητής υποδιαστολής)				
float	32	±1.17549e-038	± 3.40282e+038	%f
double	64	±2.22507e-308	± 1.79769e+308	%lf

Παράρτημα III - Μορφοποίηση κειμένου για έξοδο (printf)

Στη μορφοποιημένη έξοδο με τις εντολές `printf`, `sprint`, `fprint`, μέσα στο κείμενο μορφοποίησης επιτρέπονται διάφορες ακολουθίες που ξεκινούν με τον χαρακτήρα % ακολουθούμενο τουλάχιστον από ένα χαρακτήρα που επεξηγεί τι αναμένεται σε εκείνη τη θέση. Η γενική μορφή που μπορεί να βρεθεί εκεί είναι:

`%`*`[flags]`*`[width]`*`[.precision]`*`[length]specifier`

Οι αγκύλες εδώ δεν πληκτρολογούνται, αλλά υποδεικνύουν ότι το κάθε ένα από αυτά τα τμήματα είναι προαιρετικά.

Το `width` είναι πάντα ο χώρος (μετρημένος σε χαρακτήρες) που προορίζεται για την παράσταση της οποιασδήποτε τιμής. Εάν δεν επαρκεί, τότε δεν περικόπτεται η τιμή.

Το `precision` είναι ο χώρος οποίος θα πιάσει η τιμή (πόσα ψηφία για ακέραιους ή κείμενα) θα εκτυπωθούν. Μόνο για τους δεκαδικούς κινητής υποδιαστολής είναι ο αριθμός των ψηφίων μετά την υποδιαστολή.

Τα πιο συνηθισμένα και χρήσιμα specifier παρουσιάζονται στον παρακάτω πίνακα.

Ακέραιες τιμές

Για την επεξήγηση `flags`, εάν είναι +, τότε θα εμφανίζεται το πρόσημο και για τους θετικούς αριθμούς, εάν είναι - στοιχίζεται αριστερά η τιμή στον διαθέσιμο χώρο, εάν είναι 0 τότε ο διαθέσιμος χώρος γεμίζει με μηδενικά.

Για την επεξήγηση width.precision είναι ο χώρος (σε χαρακτήρες) που θα εκτυπωθεί η τιμή ακολουθούμενος από τον χώρο που θα καταλάβει η ίδια η τιμή.

Για την επεξήγηση length, εάν μετά το `%` υπάρχει επιπλέον το γράμμα `l` τότε η τιμή αναμένεται να είναι `long`, με το `h` αναμένεται να είναι `short` και με το `ll` αναμένεται να είναι `long long`.

Παράδειγμα: το `printf("<%+08.5d>\n",12);` *εμφανίζει 5ψήφιο αριθμό σε χώρο 8 χαρακτήρων μετρώντας και το πρόσημο. Τα σύμβολα < και > υπάρχουν για να είναι εμφανή τα κενά.*

```
<  +00012>
```

`%d ή %i`	Εκτυπώνεται η τιμή ως προσημασμένη ακέραια στο δεκαδικό.
`%u`	Εκτυπώνεται η τιμή ως απρόσημη ακέραια στο δεκαδικό σύστημα.
`%o`	Εκτυπώνεται η τιμή ως προσημασμένη ακέραια στο οκταδικό.
`%x ή %X`	Εκτυπώνεται η τιμή ως προσημασμένη ακέραια στο δεκαεξαδικό (με κεφαλαία %X ή πεζά %x).

<table>
<tr><td colspan="2">Δεκαδικές τιμές

Για την επεξήγηση <code>flags</code>, εάν είναι +, τότε θα εμφανίζεται το πρόσημο και για τους θετικούς αριθμούς, εάν είναι - στοιχίζεται αριστερά η τιμή στον διαθέσιμο χώρο, εάν είναι 0 τότε ο διαθέσιμος χώρος γεμίζει με μηδενικά.

Για την επεξήγηση width.precision είναι ο χώρος (σε χαρακτήρες) που θα εκτυπωθεί η τιμή ακολουθούμενος από τον αριθμό των ψηφίων μετά την υποδιαστολή.

Για την επεξήγηση length, εάν μετά το <code>%</code> υπάρχει επιπλέον το γράμμα <code>L</code> τότε η τιμή αναμένεται να είναι <code>double</code>, αλλιώς θεωρείται ως <code>float</code>.

Παράδειγμα: το <code>printf("<%+012.4f>\n",12.23);</code> εμφανίζει αριθμό με 4 δεκαδικά ψηφία σε χώρο 12 χαρακτήρων μετρώντας και το πρόσημο. Τα σύμβολα < και > υπάρχουν για να είναι εμφανή τα κενά.
<code><+000012.2300></code></td></tr>
<tr><td><code>%e (ή %E)</code></td><td>Εκτυπώνεται η τιμή του αριθμού κινητής υποδιαστολής σε «επιστημονική»/εκθετική μορφή χρησιμοποιώντας το e (ή E) ως διαχωριστικό της βάσης και του εκθέτη</td></tr>
<tr><td><code>%f</code></td><td>Εκτυπώνεται η τιμή του αριθμού κινητής υποδιαστολής σε απλή μορφή</td></tr>
<tr><td><code>%g (ή %G)</code></td><td>Εκτυπώνει τη συντομότερη μορφή από της %f και %e (ή %E).</td></tr>
<tr><td colspan="2">Κείμενα

Για την επεξήγηση <code>flags</code>, εάν είναι - στοιχίζεται αριστερά η τιμή στον διαθέσιμο χώρο, εάν είναι 0 τότε ο διαθέσιμος χώρος γεμίζει με μηδενικά.

Παράδειγμα: το <code>printf("<%8.4s>\n","abcdefgh");</code> εμφανίζει 4 χαρακτήρες του κειμένου σε χώρο 8 χαρακτήρων. Τα σύμβολα < και > υπάρχουν για να είναι εμφανή τα κενά.
<code>< abcd></code></td></tr>
<tr><td><code>%c</code></td><td>Εκτυπώνεται η τιμή ως χαρακτήρας βάσει της κωδικοποίησης ASCII</td></tr>
<tr><td><code>%s</code></td><td>Εκτυπώνεται τιμή ως κείμενο μέχρι τον χαρακτήρα με κωδικό 0.</td></tr>
<tr><td colspan="2">Διάφορα</td></tr>
<tr><td><code>%n</code></td><td>Δεν εκτυπώνει την αντίστοιχη τιμή</td></tr>
<tr><td><code>%p</code></td><td>Εκτυπώνει την τιμή ως δείκτη</td></tr>
<tr><td><code>%%</code></td><td>Δεν υπάρχει αντίστοιχη τιμή. Απλά εμφανίζεται ο ίδιος ο χαρακτήρας %</td></tr>
</table>

Παράρτημα IV - Μορφοποίηση κειμένου για είσοδο (scanf)

Στη μορφοποιημένη έξοδο με τις εντολές `scanf`, `sscanf`, `fscanf`, μέσα στο κείμενο μορφοποίησης επιτρέπονται διάφορες ακολουθίες που ξεκινούν με τον χαρακτήρα % ακολουθούμενο τουλάχιστον από ένα χαρακτήρα που επεξηγεί τι αναμένεται σε εκείνη τη θέση. Η γενική μορφή που μπορεί να βρεθεί εκεί είναι:

`%`*`[`*`*`]`*`[`*`width`*`]`*`[`*`modifiers`*`]type`

Οι αγκύλες εδώ δεν πληκτρολογούνται, αλλά υποδεικνύουν ότι το κάθε ένα από αυτά τα τμήματα είναι προαιρετικά.

Ο αστερίσκος υποδεικνύει ότι αναμένεται να βρεθεί μία τιμή εκεί αλλά θα αγνοηθεί και δεν θα αποθηκευτεί σε μεταβλητή.

Το `width` είναι ο μέγιστος αριθμός χαρακτήρων που επιτρέπεται να διαβαστεί.

Τα `modifiers` με το `type` παρουσιάζονται στον ακόλουθο πίνακα.

Ακέραιες Τιμές *Τα `modifiers` μπορεί να είναι ο χαρακτήρας `h` για να υποδείξει έναν `short`, ο `l` για να υποδείξει έναν `long` αντί απλού `int`, και το `ll` για να υποδείξει έναν `long long`.*	
`%d`	Τιμή εκφρασμένη στο δεκαδικό σύστημα αρίθμησης
`%u`	Απρόσημη τιμή εκφρασμένη στο δεκαδικό σύστημα αρίθμησης
`%o`	Τιμή εκφρασμένη στο οκταδικό σύστημα αρίθμησης
`%x ή %X`	Τιμή εκφρασμένη στο δεκαεξαδικό σύστημα αρίθμησης
`%i`	Τιμή εκφρασμένη σε σύστημα αρίθμησης βάσει του προθέματος (0 = οκταδικό, 0x δεκαεξαδικό, αλλιώς δεκαδικό)
Δεκαδικές τιμές *Τα `modifiers` μπορεί να είναι ο χαρακτήρας `L` για να υποδείξει έναν `double`.*	
`%e ή %E ή %f ή %g ή %G`	Προσημασμένη τιμή με δεκαδικό μέρος, στο δεκαδικό με υποδιαστολή και προαιρετικά εκθέτη διαχωριζόμενο με το e ή το E
Κείμενα	
%c	Τιμή που αντιστοιχεί σε έναν χαρακτήρα
%s	Κείμενο μέχρι να συναντηθεί ένας κενός (whitespace) χαρακτήρας

ΣΗΜΕΙΩΣΕΙΣ