ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΣΗΜΕΙΩΣΕΙΣ ΓΙΑ ΤΗ ΓΛΩΣΣΑ “C”

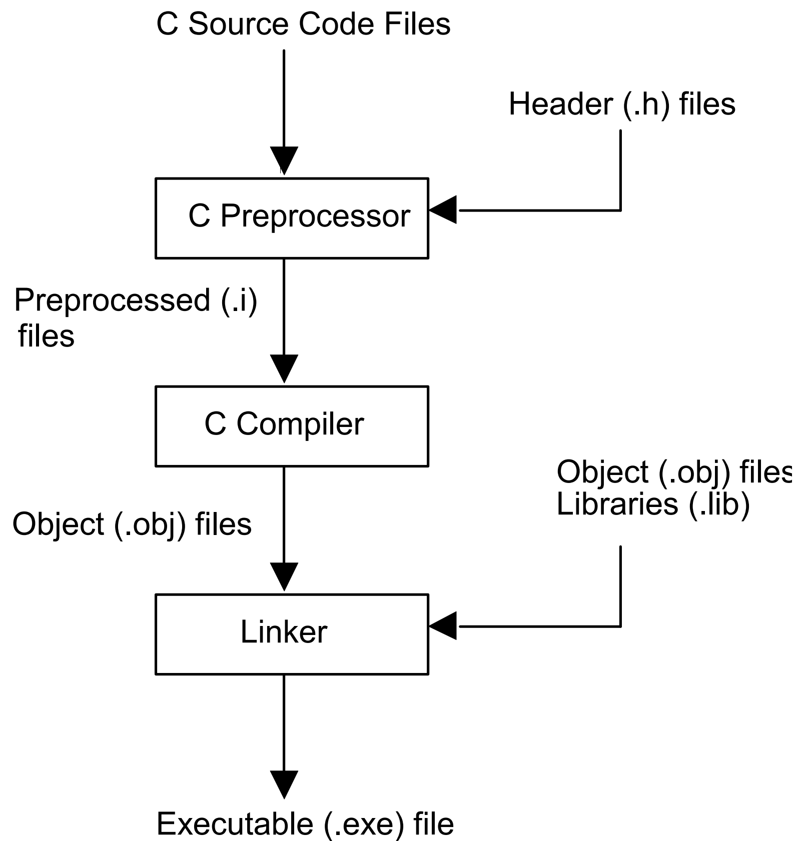
Κ. ΒΑΣΙΛΑΚΗΣ – Γ. ΛΕΠΟΥΡΑΣ

1. Συγγραφή ενός προγράμματος C

Η γλώσσα προγραμματισμού "C" είναι σχεδιασμένη για να τρέχει κάτω από όλα τα λειτουργικά συστήματα, και έτσι δεν βασίζεται σε κάποιες ειδικές ευκολίες που μπορεί να παρέχει το λειτουργικό σύστημα. Για να γράψουμε έτσι ένα πρόγραμμα C απαιτούνται:

1) ένας text editor στον οποίο θα γράψουμε το πρόγραμμά μας. Τέτοιοι editors υπάρχουν σε όλα τα λειτουργικά συστήματα
2) ο μεταγλωττιστής της γλώσσας C που θα αναλάβει να μεταφράσει το πρόγραμμά μας σε κώδικα μηχανής. Ο μεταγλωττιστής της C αποτελείται από τον C-preprocessor και τον C-compiler. Στα επόμενα με τον όρο "compiler" θα αναφερόμαστε και στα δύο αυτά τμήματα, εκτός αν καθορίζεται διαφορετικά. Ο C-preprocessor επεξεργάζεται τα αρχεία που περιέχουν κώδικα C, με τρόπο που θα αναλυθεί σε επόμενο κεφάλαιο, και παράγει ενδιάμεσα αρχεία, από τα οποία ο C-compiler παράγει αρχεία με κώδικα μηχανής (.obj) αρχεία
3) ο linker, που είναι κομμάτι του λειτουργικού συστήματος, αν και κάποια υλοποίηση ενός C-compiler μπορεί να παρέχει τον δικό της. Ο linker αναλαμβάνει να συνδέσει τον κώδικα μηχανής του προγράμματός μας (αυτόν που βρίσκεται στα .obj αρχεία) με κώδικα που υπάρχει σε βιβλιοθήκες (αρχεία .lib) και άλλα αρχεία με κώδικα μηχανής (.obj), παράγοντας τελικά ένα εκτελέσιμο αρχείο. Ο κώδικας που βρίσκεται στις βιβλιοθήκες συμπεριλαμβάνει κώδικα που υλοποιεί υψηλού επιπέδου λειτουργίες μέσα από διαδικασίες πιο χαμηλού επιπέδου, απαλλάσσοντάς μας έτσι από την υποχρέωση να κάνουμε εμείς την υλοποίηση αυτή.

Η όλη διαδικασία παραγωγής ενός εκτελέσιμου αρχείου από τα .c, .obj και .lib αρχεία φαίνεται στο πιο κάτω σχήμα.



Στα περισσότερα λειτουργικά συστήματα δεν είναι απαραίτητο να ασχοληθούμε με τη διαδικασία αυτή μια και υπάρχει κάποιο πρόγραμμα που την αναλαμβάνει. Σε μερικά μάλιστα συστήματα και ο text editor παρέχεται σε ένα ολοκληρωμένο περιβάλλον, στο οποίο μπορούμε να γράψουμε τα προγράμματα μας, να τα μεταφράσουμε και να τα συνδέσουμε, παράγοντας έτσι το εκτελέσιμο αρχείο, χωρίς να χρειαστεί να ασχοληθούμε καθόλου με την ενδιάμεση διαδικασία.

2. Τι είναι ένα πρόγραμμα C

'Ενα πρόγραμμα C αποτελείται από *οδηγίες προς τον preprocessor* (preprocessor directives), από *δηλώσεις δεδομένων* (data declarations) και από *συναρτήσεις* (functions). Οι συναρτήσεις δεν πρέπει να συγχέονται με τις συναρτήσεις που συναντάμε στα μαθηματικά· στην πραγματικότητα πρόκειται για κομμάτια κώδικα, δηλαδή για ακολουθίες εντολών της C και, πιθανώς, κάποιες δηλώσεις δεδομένων, τα οποία έχουν όνομα, μπορούν να δέχονται παραμέτρους και να επιστρέφουν αποτελέσματα. Τα ονόματα των συναρτήσεων θα πρέπει να είναι διαφορετικά μεταξύ τους, έτσι ώστε να είναι σαφές σε ποια συνάρτηση αναφερόμαστε όταν χρησιμοποιούμε κάποιο όνομα, ενώ μία από τις συναρτήσεις θα πρέπει να έχει το όνομα *main* και αντιπροσωπεύει το σημείο από το οποίο ξεκινάει και τελειώνει η εκτέλεση του προγράμματός μας. Με άλλα λόγια, η πρώτη εντολή στη συνάρτηση main είναι η πρώτη εντολή του προγράμματος που θα εκτελεστεί, ενώ με την εκτέλεση της τελευταίας εντολής της συνάρτησης main τελειώνει και η εκτέλεση του προγράμματος. Σε ένα πρόγραμμα C μπορούμε να έχουμε και σχόλια: Η αρχή των σχολίων δηλώνεται με "/*" και το τέλος τους με "*/". Οτιδήποτε βρίσκεται ανάμεσα από την αρχή και το τέλος των σχολίων αγνοείται.

Οι τύποι δεδομένων στη C

Η C παρέχει τους πιο κάτω τύπους δεδομένων, τους οποίους μπορεί να χρησιμοποιήσει ένα πρόγραμμα:

α) *int*, *short int* και *long int* που χρησιμοποιούνται για αναπαράσταση ακεραίων. Η διαφορά ανάμεσα στους τρεις τύπους βρίσκεται στο εύρος των τιμών που καλύπτουν και στο χώρο αποθήκευσης που χρειάζονται. Τυπικά ένας ακέραιος τύπου *short int* χρειάζεται δύο bytes (16 bits) για να αποθηκευθεί και μπορεί να αναπαραστήσει τιμές ανάμεσα στο -32768 και το 32767. 'Ενας ακέραιος τύπου *long int* απαιτεί συνήθως τέσσερα bytes (32 bits) για την αποθήκευσή του και μπορεί να αναπαραστήσει τιμές στο διάστημα -2147483648 (-2^{31}) έως 2147483647 ($2^{31} - 1$). Ο τύπος *int* είναι συνήθως ισοδύναμος με κάποιον από τους δύο πιο πάνω τύπους, και το μέγεθός του είναι συνήθως ίσο με το μέγεθος της λέξης του υπολογιστή. Αν κάποια ποσότητα που θέλουμε να αναπαραστήσουμε παίρνει μόνο θετικές τιμές τότε μπορούμε να το δηλώσουμε, με αποτέλεσμα με τον ίδιο χώρο αποθήκευσης να έχουμε μεγαλύτερο εύρος τιμών στην επιθυμητή περιοχή. Η δήλωση γίνεται με τη λέξη *unsigned*. 'Ετσι, ο τύπος δεδομένων *unsigned short int* αντιπροσωπεύει τιμές στο διάστημα [0, 65535], ο τύπος *unsigned long int* μπορεί να πάρει τιμές από 0 έως 4294967295 ($2^{32} - 1$)

β) *char*, που χρησιμοποιείται για την αποθήκευση χαρακτήρων. Στην πραγματικότητα η C χειρίζεται τους χαρακτήρες σαν ακεραίους των 8 bits (1 byte) που παίρνουν τιμές από -128 έως 127, καλύπτοντας έτσι τον κώδικα ASCII που χρειάζεται τιμές από 0 έως 127. Υπάρχει και ο τύπος *unsigned char* που παίρνει τιμές από 0 έως 255

γ) *float* και *double* που χρησιμοποιούνται για την αναπαράσταση πραγματικών αριθμών. Ο τύπος *float* χρειάζεται συνήθως τέσσερα bytes για να αποθηκευθεί, και αναπαριστά τιμές από $1.5 * 10^{-45}$ έως $3.4 * 10^{38}$ (και τους αντίστοιχους αρνητικούς), με δυνατότητα αποθήκευσης 7 σημαντικών ψηφίων, ενώ ο τύπος *double* απαιτεί οκτώ bytes για την αποθήκευση του, αναπαριστώντας συνήθως τιμές από $5 * 10^{-324}$ μέχρι $1.7 * 10^{308}$, με δυνατότητα αναπαράστασης 15 σημαντικών ψηφίων. Μερικοί compilers υποστηρίζουν και τον τύπο δεδομένων

long double που απαιτεί 10 bytes για την αποθήκευσή του, αναπαριστά τιμές από $3.4 * 10^{-4932}$ μέχρι $1.1 * 10^{4932}$ και δίνει ακρίβεια 19 σημαντικών ψηφίων

δ) *δείκτες (pointers)* σε δεδομένα των πιο πάνω τύπων ή σε *void*. Οι δείκτες και η χρήση τους θα αναλυθούν σε επόμενο κεφάλαιο.

Η C δεν διαθέτει τύπο που να αναπαριστά *λογικές μεταβλητές* (π.χ. Boolean στην Pascal). Οι λογικές παραστάσεις στη C είναι στην πραγματικότητα ακέραιοι (char, short int , int ή long) και θεωρούνται αληθείς όταν η τιμή τους είναι διαφορετική από το μηδέν και ψευδείς όταν η τιμή τους είναι μηδέν.

Οι πιο πάνω τύποι είναι οι μόνοι που υποστηρίζονται άμεσα από τη C. Ο χρήστης (ή ο προμηθευτής του compiler) μπορεί χρησιμοποιώντας τους τύπους αυτούς να ορίσει πιο σύνθετους.

Δηλώσεις σταθερών στη C

Σε διάφορα σημεία ενός προγράμματος C θα χρειαστεί να ορίσουμε σταθερές. Οι σταθερές μπορεί να είναι οποιουδήποτε τύπου από τους προαναφερθέντες, καθώς και *strings*, δηλαδή συμβολοσειρών (περισσότερα για τα strings θα αναπτυχθούν στη συνέχεια).

Για να δηλώσουμε μία ακέραια σταθερά χρησιμοποιούμε τους κλασικούς αραβικούς συμβολισμούς. 'Ετσι για να δηλώσουμε τη σταθερά ένα γράφουμε *1*, ενώ ο συμβολισμός *1207* δηλώνει τον αριθμό χίλια διακόσια επτά. Αν επιθυμούμε μία ακέραια σταθερά να είναι μη προσημασμένη, παραθέτουμε στο τέλος της το χαρακτήρα *U* ή το χαρακτήρα *u*. 'Ετσι ο συμβολισμός *40000U* αναπαριστά τη μη προσημασμένη σταθερά σαράντα χιλιάδες. Αν θέλουμε μία ακέραια σταθερά να είναι τύπου *long int* παραθέτουμε στο τέλος της το χαρακτήρα *L* ή το χαρακτήρα *l*. Π.χ. ο συμβολισμός *70000l* αναπαριστά τη σταθερά εβδομήντα χιλιάδες με τύπο *long int*. Μπορούμε βέβαια να δηλώσουμε και σταθερές τύπου *unsigned long int* παραθέτοντας τους χαρακτήρες *LU*.

Οι σταθερές δηλωμένες κατ' αυτόν τον τρόπο εκλαμβάνονται ως αριθμοί του δεκαδικού συστήματος αρίθμησης. Μπορούμε να ορίσουμε σταθερές επίσης:

α) στο οκταδικό σύστημα αρίθμησης, γεγονός που επιτυγχάνεται με το να είναι το *0* πρώτο ψηφίο της σταθεράς. 'Ετσι η σταθερά *010* ΔΕΝ είναι ο αριθμός 10 του δεκαδικού συστήματος αρίθμησης, αλλά ο αριθμός 8. Στο οκταδικό σύστημα αρίθμησης δεν μπορούν να χρησιμοποιηθούν τα ψηφία 8 και 9

β) στο δεκαεξαδικό σύστημα αρίθμησης, βάζοντας πριν από τη σταθερά τους χαρακτήρες *0x* ή *0X*. Ως ψηφία ενός δεκαεξαδικού αριθμού μπορούν να χρησιμοποιηθούν τα ψηφία *0* έως *9* και οι χαρακτήρες *aAbBcCdDeEfF*.

Για να δηλώσουμε μία σταθερά τύπου χαρακτήρα (char) θα χρησιμοποιήσουμε το συμβολισμό 'x', όπου x μπορεί να είναι οποιοσδήποτε χαρακτήρας του ASCII αλφάβητου και ο συμβολισμός θα αντιστοιχεί στο χαρακτήρα αυτό. Επειδή όμως μερικοί χαρακτήρες δεν μπορούν να γραφούν από το πληκτρολόγιο ή έχουν ειδική σημασία για τον compiler ή είναι διαφορετικοί ανάμεσα σε υπολογιστικά συστήματα, έχουμε το δικαίωμα να ορίσουμε σταθερές τύπου χαρακτήρα και με τους ακόλουθους δύο τρόπους:

α) μέσω *escape sequences*, όπως π.χ. τους συμβολισμούς '\n', '\t', '\\'f2, '\?' '\'f2' και '\"' που δηλώνουν το newline, το tab, και τους χαρακτήρες \, *?*, *'* και *"* αντίστοιχα

β) μέσω των κωδικών τους στο αλφάβητο ASCII που μπορούν να δοθούν είτε στο οκταδικό είτε στο δεκαεξαδικό σύστημα. 'Ετσι ο συμβολισμός *'\040'* αντιστοιχεί στο χαρακτήρα με κωδικό τον οκταδικό αριθμό *040* και ο συμβολισμός *'\x41'* συμβολίζει το χαρακτήρα με κωδικό τον δεκαεξαδικό αριθμό *41*.

Μία πραγματική σταθερά μπορεί να δοθεί κατά δύο τρόπους:

α) με τον κλασικό συμβολισμό όπου η τελεία χωρίζει το ακέραιο μέρος από το πραγματικό μέρος, π.χ. 1.3, -300.02

β) αν η απόλυτη τιμή του αριθμού είναι μεγάλη (π.χ. 10^{30}) ή πολύ μικρή (π.χ. 10^{-32}), τότε δεν είναι βολικό να γράφουμε όλα τα ψηφία του· αντί γι' αυτό μπορούμε να δώσουμε ένα πλήθος σημαντικών ψηφίων και τη δύναμη του 10 με την οποία πολλαπλασιάζεται ο αριθμός. Αυτό γίνεται με το συμβολισμό:
αααα.δδδδδ[eE]εκθέτης
όπου *αααα* είναι ακέραια ψηφία, *δδδδ* είναι δεκαδικά ψηφία και *εκθέτης* είναι η δύναμη του 10 (ενδεχομένως αρνητική) με την οποία θα πολλαπλασιαστεί ο αριθμός. Τα ακέραια ή τα δεκαδικά ψηφία (όχι όμως και τα δύο) μπορούν να παραλειφθούν. 'Ετσι ο συμβολισμός ***1e10*** συμβολίζει τον αριθμό 10^{10}, ο συμβολισμός 303.26e-25 τον αριθμό 3.02326 * 10^{-23} και ο αριθμός -.892e+9 τον αριθμό -8.92 * 10^{8}.

Και οι δύο μορφές πραγματικών που δηλώθηκαν πιο πάνω υποδηλώνουν σταθερές τύπου *double*. Για να δηλώσουμε ότι μία σταθερά είναι τύπου *float* τη γράφουμε όπως στα πιο πάνω και παραθέτουμε στο τέλος της το χαρακτήρα *f* ή το χαρακτήρα *F*. Αντίστοιχα, η παράθεση μετά το τέλος της σταθεράς του χαρακτήρα *l* ή του χαρακτήρα *L* μετατρέπει τη σταθερά σε τύπου *long double*.

Οι σταθερές τύπου string περιέχουν συνήθως μηνύματα που θα τυπωθούν. Για να δηλώσουμε μία σταθερά τύπου string περικλείουμε τους χαρακτήρες που θα αποτελούν το μήνυμα με τον χαρακτήρα *"*. Μπορούμε στη θέση ενός χαρακτήρα να χρησιμοποιήσουμε τα escape sequences και τις οκταδικές και δεκαεξαδικές αναπαραστάσεις που είδαμε στις σταθερές τύπου χαρακτήρα. 'Ετσι αν τυπωθεί η σταθερά "Hello there\nBye\x41" η οθόνη μας θα δείξει:

```
Hello there
ByeA
```

γιατί το escape sequence *\n* αντιστοιχεί στο newline, και ο χαρακτήρας με κωδικό το δεκαεξαδικό 41 στο αλφάβητο ASCII είναι ο A.

Δηλώσεις μεταβλητών στη C

'Οπως σε κάθε γλώσσα προγραμματισμού έτσι και στη C έχουμε το δικαίωμα να ορίσουμε μεταβλητές, δηλαδή ονόματα που αντιπροσωπεύουν κάποια τιμή. Η τιμή αυτή μπορεί να αλλάζει με το χρόνο, δηλαδή κάποια στιγμή μια μεταβλητή να έχει τιμή A και σε κάποιο μεταγενέστερο χρονικό σημείο να έχει τιμή B. Κάθε μεταβλητή, κάθε όνομα δηλαδή, έχει έναν *τύπο*, ο οποίος είναι ο ίδιος καθ' όλη τη διάρκεια ζωής της μεταβλητής, και τόσο το όνομά της όσο και ο τύπος της πρέπει να έχουν δηλωθεί *πριν* χρησιμοποιηθεί η μεταβλητή για πρώτη φορά.

Για να δηλώσουμε μία μεταβλητή με όνομα A που να είναι τύπου X (ο οποίος μπορεί να είναι ένας από τους προαναφερθέντες ή κάποιος που έχει ορισθεί από το χρήστη ή σε κάποια βιβλιοθήκη) χρησιμοποιούμε το συμβολισμό:

X A;

ενώ σε μία δήλωση μπορούμε να ορίσουμε περισσότερες από μία μεταβλητές του ίδιου τύπου. 'Ετσι με τη δήλωση:

X A, B;

ορίζουμε τις μεταβλητές A και B που είναι τύπου X. Μία μεταβλητή που έχει ορισθεί κατ' αυτόν τον τρόπο θα έχει αρχικά κάποια *τυχαία* τιμή, δεν πρέπει δηλαδή να κάνουμε καμία υπόθεση για την τιμή της μεταβλητής αυτής. Αν θέλουμε να δώσουμε κάποια αρχική τιμή σε μία μεταβλητή θα πρέπει να χρησιμοποιήσουμε το συμβολισμό:

X A = T;
όπου X είναι τύπος δεδομένων, A είναι όνομα μεταβλητής και T είναι κάποια *σταθερή* τιμή με τύπο ίδιο με το X.
Τα ονόματα των μεταβλητών αποτελούνται από γράμματα, αριθμούς και τον χαρακτήρα "_" (underscore) και πρέπει να αρχίζουν από γράμμα ή "_", αλλά η χρήση του "_" ως πρώτου χαρακτήρα θα πρέπει να αποφεύγεται. Μπορεί να έχουν οποιοδήποτε μήκος, αν και μόνο οι πρώτοι τριάντα ένας χαρακτήρες είναι εγγυημένο ότι θα λαμβάνονται υπ' όψιν (δηλαδή ονόματα που διαφέρουν στον τριακοστό δεύτερο χαρακτήρα είναι δυνατό να εκληφθούν ως όμοια). Τα κεφαλαία και τα μικρά είναι διαφορετικά γράμματα, δηλαδή τα ονόματα:

```
myvariable
MYVARIABLE
MyVariable
```

είναι όλα διαφορετικά μεταξύ τους.

Οι εντολές της C

Η C παρέχει ένα πολύ μικρό σύνολο από εντολές, και αφήνει τις περισσότερες λειτουργίες που είναι απαραίτητες σε ένα πρόγραμμα, όπως π.χ. είσοδος/έξοδος, να υλοποιούνται μέσα από βιβλιοθήκες. Αυτό γίνεται για να είναι εύκολη η υλοποίηση του compiler και για να εξασφαλίζεται πιο εύκολα η συμβατότητα και η μεταφερτότητα ανάμεσα στα διάφορα υπολογιστικά συστήματα. Οι εντολές που παρέχει η C είναι οι ακόλουθες:

```
break
continue
do
for
goto
if
return
switch
while
```

Η σύνταξη και η λειτουργία των εντολών αυτών θα αναλυθεί σε επόμενα κεφάλαια.

3. 'Ενα απλό πρόγραμμα C

Στο σχήμα 1 φαίνεται ένα απλό πρόγραμμα C το οποίο τυπώνει στην οθόνη ένα φιλικό μήνυμα:

```
#include <stdio.h>

main()
{
      printf("Greetings from your first C program\n");
}
```

Σχήμα 1

'Οπως βλέπουμε το πρόγραμμα αποτελείται από τα εξής συστατικά στοιχεία:

α) μία οδηγία προς τον preprocessor (*#include <stdio.h>)* να συμπεριλάβει στο σημείο εκείνο τα περιεχόμενα του αρχείου *stdio.h*, το οποίο βρίσκεται σε κάποιο προκαθορισμένο για τον compiler directory. Το αρχείο αυτό περιέχει χρήσιμες για τον compiler πληροφορίες που αφορούν την είσοδο/έξοδο

β) τη συνάρτηση *main()* η οποία θα πρέπει να υπάρχει σε κάθε C πρόγραμμα. Η συνάρτηση main() του πιο πάνω προγράμματος περιέχει μία μόνο μία κλήση στη συνάρτηση *printf()*, η οποία είναι τμήμα της βασικής βιβλιοθήκης εισόδου/εξόδου της C και αναλαμβάνει να τυπώσει στην οθόνη σταθερές και δεδομένα κατά τον τρόπο που εμείς θα καθορίσουμε. Η συγκεκριμένη κλήση τυπώνει τη σταθερά τύπου string *"Greetings from your first C program\n"* (όπως έχουμε πει το *\n* αντιστοιχεί στο newline) που βρίσκεται ανάμεσα στις παρενθέσεις που ακολουθούν το όνομα printf. Μια και η κλήση αυτή είναι και η τελευταία εντολή της συνάρτησης main(), μετά το τύπωμα του string το πρόγραμμα τερματίζει.

Στο πιο πάνω πρόγραμμα βλέπουμε και τον τρόπο με τον οποίο γράφουμε δικές μας συναρτήσεις στη C (ή, καλύτερα, το ελάχιστο που απαιτείται για μία δική μας συνάρτηση): πρέπει να δώσουμε το όνομα της συνάρτησης, το οποίο πρέπει να είναι μοναδικό - δεν πρέπει δηλαδή να υπάρχει άλλη συνάρτηση με το ίδιο όνομα - κατόπιν να βάλουμε δύο παρενθέσεις ("*()*") που δείχνουν στη C ότι δηλώνουμε συνάρτηση και όχι δεδομένο, κατόπιν το χαρακτήρα "*{*" που δηλώνει την αρχή των εντολών που υλοποιούν τη συνάρτηση, στη συνέχεια μία ή περισσότερες εντολές και, τέλος, το χαρακτήρα "*}*" που δηλώνει το τέλος της συνάρτησης. Προσοχή στο ότι δεν υπάρχει ο χαρακτήρας "*;*" μετά από τις παρενθέσεις που ακολουθούν το όνομα της συνάρτησης την οποία δηλώνουμε.

Η εκτέλεση των εντολών που περιέχει μία συνάρτηση γίνεται από την πρώτη εντολή προς την τελευταία. Η εκτέλεσή της τερματίζεται όταν δεν υπάρχει άλλη εντολή ή όταν εκτελεστεί η εντολή *return*. Αν εκτελεστεί η εντολή return, τότε δεν εκτελούνται οι υπόλοιπες εντολές της συνάρτησης (αυτές που βρίσκονται μετά το return).

Δήλωση συναρτήσεων στη C

Πιο πάνω είδαμε με τη συνάρτηση *main()* το ελάχιστο που απαιτείται για να δηλώσουμε μία συνάρτηση C. Μία συνάρτηση όμως μπορεί να είναι πολύ πιο πολύπλοκη: Μπορεί να δέχεται παραμέτρους, να επιστρέφει κάποιο αποτέλεσμα, να έχει τοπικά δεδομένα, κ.τ.λ. Η γενική μορφή μίας συνάρτησης C είναι η ακόλουθη:

```
[τύπος δεδομένου] όνομα_συνάρτησης(λίστα παραμέτρων)
[δηλώσεις τύπων παραμέτρων]
{
      [δηλώσεις τοπικών μεταβλητών]
      εντολές
```

}
(ο συμβολισμός [x] σημαίνει ότι το τμήμα x είναι προαιρετικό). Το *όνομα_συνάρτησης* θα πρέπει να είναι μοναδικό και ο *τύπος δεδομένου* είναι κάποιος τύπος δεδομένου (άμεσα υποστηριζόμενου από τη γλώσσα ή ορισμένος από το χρήστη) που επιστρέφεται από τη συνάρτηση· όταν παραλείπεται η C υποθέτει ότι η συνάρτηση επιστρέφει ακέραιο, ενώ αν κάποια συνάρτηση δεν θέλουμε να επιστρέφει καμία τιμή τότε στη θέση του *τύπου δεδομένου* βάζουμε τη λέξη *void*. Η *λίστα παραμέτρων* αποτελείται από μηδέν ή περισσότερα ονόματα δεδομένων με τα οποία μπορούμε να αναφερθούμε σε τιμές που προμήθευσε στην συνάρτηση αυτή η συνάρτηση που την κάλεσε (η κλήση συναρτήσεων θα αναλυθεί πιο κάτω), και για κάθε παράμετρο θα πρέπει να υπάρχει μία δήλωση τύπου παραμέτρου ανάμεσα από τη γραμμή με τις παραμέτρους και την αριστερή αγκύλη (*{*). Η δηλώσεις τύπων παραμέτρων έχουν την ίδια ακριβώς μορφή με τις δηλώσεις δεδομένων, εκτός από το ότι δεν είναι δυνατό να έχουμε αρχικοποιήσεις. Μπορούμε να συμπτύξουμε πολλές δηλώσεις σε μία, αρκεί οι παράμετροι να έχουν τον ίδιο τύπο. Μέσα στη συνάρτηση μπορούμε να έχουμε δηλώσεις δεδομένων, τα οποία είναι τοπικά για τη συνάρτηση, και δεν έχουν *καμία απολύτως σχέση* με δεδομένα που έχουν ορισθεί σε άλλες συναρτήσεις, *έστω και αν έχουν το ίδιο όνομα*. Τα δεδομένα αυτά μπορούν να χρησιμοποιούνται μόνο μέσα στη συνάρτηση αυτή. Οι εντολές, τέλος, έχουν την ίδια σημασία με αυτή που περιγράφηκε για τη συνάρτηση main(). Αν η συνάρτηση επιστρέφει κάποιο τύπο δεδομένων (αν δηλαδή στη θέση του *τύπου δεδομένου* δεν είναι η λέξη void) τότε η συνάρτηση θα πρέπει να τερματίζει την εκτέλεσή της με μία εντολή
return έκφραση;
εντολή που έχει ως αποτέλεσμα να επιστρέφεται στη συνάρτηση που κάλεσε την τρέχουσα η τιμή της έκφρασης. Ο τύπος της έκφρασης θα πρέπει να συμφωνεί με τον *τύπο δεδομένου*.
Στο σχήμα 2 φαίνεται ένα απλό πρόγραμμα C που αποτελείται από δύο συναρτήσεις: Τη συνάρτηση main και τη συνάρτηση sum. Η συνάρτηση main() δεν δέχεται παραμέτρους και επιστρέφει ακέραιο, ενώ η συνάρτηση sum δέχεται ως παραμέτρους δύο ακεραίους και επιστρέφει το άθροισμά τους, το οποίο είναι επίσης ακέραιος.

```
#include <stdio.h>

int main()
{
      int result;
      result = sum(3, 8);
      printf("%d + %d = %d\n", 3, 8, result);
      return 0;
}

int sum(a, b)
int a, b; /* δηλώνουμε ότι τα a και b είναι ακέραιοι */
{
      return a + b; /* επιστρέφεται το άθροισμα */
}
```

Σχήμα 2

Στη συνάρτηση main() του σχήματος 2 παρατηρούμε ότι έχουμε το δικαίωμα να πάρουμε το αποτέλεσμα μίας συνάρτησης που επιστρέφει κάποιο τύπο δεδομένων και να το χρησιμοποιήσουμε όπως θα χρησιμοποιούσαμε και μία σταθερά του τύπου

αυτού. Στο παράδειγμά μας, το αποτέλεσμα της συνάρτησης sum(), το οποίο είναι ακέραιος, εκχωρείται (μέσω του τελεστή =) σε μία μεταβλητή του ίδιου τύπου. Η μεταβλητή αυτή θα έχει ως τιμή, αμέσως μετά την εκτέλεση της εκχώρησης, το αποτέλεσμα της συνάρτησης sum(3, 8), που είναι φυσικά 11.

'Οταν σε κάποιο σημείο ενός προγράμματος ο compiler βρίσκει το όνομα μίας συνάρτησης ακολουθούμενο από παρενθέσεις, οι οποίες μπορεί να περιέχουν και παραμέτρους, δημιουργεί μία κλήση στη συνάρτηση αυτή: Ο έλεγχος μεταφέρεται προσωρινά στη συνάρτηση της οποίας το όνομα βρέθηκε, εκτελούνται οι εντολές της συνάρτησης αυτής και όταν αυτές τελειώσουν, ή εκτελεστεί μία εντολή return, τότε ο έλεγχος επανέρχεται στη συνάρτηση μέσα στην οποία υπήρχε η κλήση, και αυτή συνεχίζει κανονικά από το σημείο της κλήσης. Μία συνάρτηση που κλήθηκε έχει το δικαίωμα να καλέσει και άλλες συναρτήσεις, κ.ο.κ. Στο παράδειγμά μας, η ύπαρξη του συμβολισμού

```
sum(8, 3)
```

δημιουργεί μία κλήση στη συνάρτηση sum, και τα a και b παίρνουν τις τιμές των αντιστοίχων παραμέτρων (3 και 8 αντίστοιχα). Κατόπιν εκτελείται η εντολή

```
return a + b;
```

με αποτέλεσμα να καθορίζεται ότι η τιμή που επιστρέφει η συνάρτηση είναι το a + b, (8 + 3 = 11) και να επιστρέφεται ο έλεγχος στη συνάρτηση main().

Η επόμενη γραμμή της συνάρτησης main() περιέχει μία κλήση στη συνάρτηση printf(), με μία πιο γενική σύνταξη από αυτή που έχουμε δει ως τώρα. Πιο συγκεκριμένα, βλέπουμε ότι η printf() παίρνει σαν παραμέτρους μία σταθερά τύπου string (μπορεί να είναι και μεταβλητή) και τρεις ακεραίους τους οποίους και εκτυπώνει. Στην πραγματικότητα η printf() μπορεί να πάρει σαν παραμέτρους μία σταθερά τύπου string η οποία ονομάζεται format και κατόπιν μία σειρά από παραμέτρους οποιουδήποτε τύπου από αυτούς που υποστηρίζει άμεσα η γλώσσα, και να τις τυπώσει σύμφωνα με τις οδηγίες που θα βρει στο format. Οι οδηγίες στο format αφορούν τον τύπο των παραμέτρων, το πλήθος χαρακτήρων που θα χρησιμοποιηθούν για την εκτύπωση κάθε παραμέτρου, τη στοίχιση, την ακρίβεια, κ.τ.λ. Πλήρης ανάλυση της printf() μπορεί να βρεθεί στο παράρτημα Α.

Το format στην printf λειτουργεί ως εξής: Για κάθε χαρακτήρα, από την αρχή ως το τέλος του, ελέγχεται αν είναι escape sequence. Αν είναι, εκτυπώνεται ο χαρακτήρας που αντιστοιχεί στο escape sequence αυτό. Αν όχι, ελέγχεται αν είναι ο χαρακτήρας %, και αν δεν είναι, τότε εκτυπώνεται. Αν είναι %, τότε οι επόμενοι χαρακτήρας μέχρι τον επόμενο προσδιοριστή τύπου θεωρούνται ως οδηγίες για την εκτύπωση της επόμενης παραμέτρου, της οποίας ο τύπος δίνεται από τον προσδιοριστή τύπου. Οι προσδιοριστές τύπου που μπορούν να χρησιμοποιηθούν είναι οι ακόλουθοι:

α) *d* ή *i* που δηλώνουν ότι η αντίστοιχη παράμετρος είναι ακέραιος (int) που θα εκτυπωθεί στο δεκαδικό σύστημα αρίθμησης. Πριν από το χαρακτήρα d (i) μπορεί να παρεμβληθεί ο χαρακτήρας h για να δηλώσει παράμετρο τύπου short int, ή ο χαρακτήρας l για να δηλώσει παράμετρο τύπου long int

β) *o* που δηλώνει ότι η επόμενη παράμετρος είναι ακέραιος που θα εκτυπωθεί στο οκταδικό σύστημα αρίθμησης ως μη προσημασμένος. Μπορούν να χρησιμοποιηθούν οι χαρακτήρες h και l όπως στο (α)

γ) *X* ή *x* που δηλώνει ότι η επόμενη παράμετρος είναι ακέραιος που θα εκτυπωθεί στο δεκαεξαδικό σύστημα αρίθμησης ως μη προσημασμένος. Μπορούν να χρησιμοποιηθούν οι χαρακτήρες h και l όπως στο (α)

δ) *u* που δηλώνει ότι η επόμενη παράμετρος είναι μη προσημασμένος ακέραιος που θα εκτυπωθεί στο δεκαδικό σύστημα αρίθμησης. Μπορούν να χρησιμοποιηθούν οι χαρακτήρες h και l όπως στο (α)

ε) *c* που δηλώνει ότι η επόμενη παράμετρος είναι τύπου χαρακτήρα. Εκτυπώνεται ο χαρακτήρας με ASCII κωδικό την τιμή της παραμέτρου.

στ) *s* που καθορίζει ότι η επόμενη παράμετρος είναι string

ζ) *f*, *e*, *E*, *g*, *G* που καθορίζουν ότι η επόμενη παράμετρος είναι τύπου double. Αν χρησιμοποιηθεί το f η παράμετρος θα εκτυπωθεί στη μορφή ααααα.δδδδδ, αν χρησιμοποιηθεί το e ή το E τότε η παράμετρος θα εκτυπωθεί στη μορφή α.δδδδe± ε και αν χρησιμοποιηθεί το g ή το G τότε η παράμετρος θα εκτυπωθεί σε μία από τις πιο πάνω μορφές, ανάλογα με τη δύναμη του 10 που αντιστοιχεί σ' αυτή. Πριν από τους χαρακτήρες αυτούς μπορεί να τοποθετηθεί ο χαρακτήρας L που δηλώνει ότι η παράμετρος είναι τύπου long double. Δεν υπάρχει προσδιοριστής για παραμέτρους τύπου float γιατί οι παράμετροι τύπου float μετατρέπονται σε double όταν πρόκειται να περάσουν ως παράμετροι σε κάποια συνάρτηση

η) *p* που δηλώνει ότι η επόμενη παράμετρος είναι τύπου pointer

θ) *%* που καθορίζει ότι απλά θα τυπωθεί ο χαρακτήρας %.

Σύμφωνα με τα πιο πάνω η κλήση της printf() στη συνάρτηση main() του σχήματος 2 καθορίζει τα ακόλουθα: Πρώτα θα εκτυπωθεί μία παράμετρος τύπου int που ακολουθεί το format (3), κατόπιν θα τυπωθεί ένα κενό, ο χαρακτήρας '+' και άλλο ένα κενό, στη συνέχεια μία παράμετρος τύπου int (8), ακολούθως ένα κενό, ο χαρακτήρας '=' και άλλο ένα κενό, και τέλος μία παράμετρος τύπου int (result) και ένα newline.

Ο προγραμματιστής είναι ο υπεύθυνος για τη διατήρηση της αντιστοιχίας ανάμεσα στους προσδιοριστές τύπων και στους τύπους των παραμέτρων. 'Ετσι, αν έχουμε την κλήση της `printf()`:

```
printf("%c%s%d\n", '>', "Hello ", "there");
```

ο compiler δεν θα διαγνώσει το σφάλμα ότι ζητάμε από την printf() να τυπώσει έναν ακέραιο (%d) ενώ περνάμε ως παράμετρο ένα string. Το πρόγραμμά μας θα μεταφραστεί κανονικά, και η printf θα τυπώσει έναν περίεργο ακέραιο στη θέση της παραμέτρου αυτής.

Το τελευταίο σημείο που θα παρατηρήσουμε στο πρόγραμμα του σχήματος 2 είναι η ύπαρξη της εντολής

```
return 0;
```

στη συνάρτηση main(). Μια και η συνάρτηση main() συνήθως δεν καλείται από καμία άλλη συνάρτηση αλλά αποτελεί το σημείο εκκίνησης και τερματισμού του προγράμματος, θα αναρωτιόταν κανείς για τη σκοπιμότητα επιστροφής κάποιας τιμής. Στην πραγματικότητα η τιμή που επιστρέφεται από τη συνάρτηση main() παραλαμβάνεται από το λειτουργικό σύστημα του υπολογιστή, και από εκεί μπορεί κανείς να ζητήσει να εξετάσει την τιμή της (π.χ. μέσω του ERRORLEVEL στο DOS ή της μεταβλητής $status στο Unix). Για το λόγο αυτό η τιμή που επιστρέφεται από το main() λέγεται κατάσταση εξόδου (exit status) του προγράμματος και πρέπει να παίρνει κατάλληλη τιμή για να αντικατοπτρίζει το λόγο τερματισμού. Συνήθως κατάσταση εξόδου 0 αντιστοιχεί σε επιτυχή εκτέλεση, ενώ ο χρήστης μπορεί να ορίσει τη σημασία που έχουν οι άλλες καταστάσεις εξόδου για το πρόγραμμά του.

4.'Ελεγχος τύπου επιστροφής και τύπων παραμέτρων συναρτήσεων

Στο παράδειγμα του σχήματος 2 είδαμε τη χρήση μιας συνάρτησης (sum()) η οποία επέστρεφε ακέραιο και έπαιρνε δύο ακέραιες παραμέτρους. Αν αλλάξουμε λίγο το πρόγραμμα αυτό σε αυτό του σχήματος 3, τότε θα διαπιστώσουμε ότι ο compiler θα διαμαρτυρηθεί, λέγοντας ότι υπάρχει ασυμφωνία ανάμεσα στη χρήση της συνάρτησης και στον ορισμό της.

```
#include <stdio.h>

int main()
{
	double result;
	result = sum(3, 8);
	printf("%f + %f = %f\n", 3.0, 8.0, result);
	return 0;
}

double sum(a, b)
double a, b; /* δηλώνουμε ότι τα a και b είναι ακέραιοι */
{
	return a + b; /* επιστρέφεται το άθροισμα */
}
```

Σχήμα 3

Αυτό συμβαίνει γιατί ο compiler βρίσκει για πρώτη φορά τη συνάρτηση sum() στη συνάρτηση main() και, μια και δεν έχει πληροφορίες γι' αυτή, θεωρεί ότι επιστρέφει ακέραιο. 'Οταν αργότερα συναντά τον ορισμό της συνάρτησης και διαπιστώνει ότι επιστρέφει double μας αναφέρει την ασυνέπεια αυτή.
Για να μπορέσουμε να παρακάμψουμε το πρόβλημα αυτό μπορούμε να κάνουμε δύο πράγματα:

α) να μετακινήσουμε την υλοποίηση της συνάρτησης sum() πριν από τη συνάρτηση main(), έτσι ώστε ο compiler να ξέρει ότι η συνάρτηση επιστρέφει double πριν χρησιμοποιηθεί για πρώτη φορά

β) να μη μετακινήσουμε την υλοποίηση της συνάρτησης, αλλά πριν τη χρησιμοποιήσουμε να πληροφορήσουμε τον compiler ότι η συνάρτηση αυτή επιστρέφει double. Αυτό γίνεται τοποθετώντας τη δήλωση:
double sum();
πριν από τη συνάρτηση main(), ή, γενικότερα, πριν χρησιμοποιηθεί για πρώτη φορά η συνάρτηση sum(). Παρατηρούμε ότι, σε αντίθεση με την υλοποίηση της συνάρτησης πρέπει να βάλουμε τη λατινική άνω τελεία (ελληνικό ερωτηματικό) μετά τη δήλωση.

Χρησιμοποιώντας οποιαδήποτε από τις δύο μεθόδους το πρόγραμμα μας θα μεταφραστεί και θα τρέξει κανονικά.
Δεν είναι σπάνιο στη C να κάνουμε λάθος κατά το πέρασμα παραμέτρων σε κάποια συνάρτηση. Τα πιο συχνά λάθη που μπορούμε να κάνουμε αφορούν το πλήθος και τον τύπο των παραμέτρων, να περάσουμε δηλαδή περισσότερες ή λιγότερες παραμέτρους σε κάποια συνάρτηση, ή να χρησιμοποιήσουμε παράμετρο λάθος τύπου. Η κλασική C δεν παρέχει κανένα μέσο ελέγχου σφαλμάτων τέτοιου τύπου, και έτσι έπρεπε να χρησιμοποιούνται άλλα προγράμματα για την ανίχνευσή τους (π.χ. *lint* στο Unix). Η C++ όμως εισήγαγε την έννοια του πρότυπου συνάρτησης, την οποία

υιοθέτησε αργότερα και η ANSI C. Τα πρότυπα συναρτήσεων είναι ένας μηχανισμός ελέγχου του πλήθους και του τύπου των παραμέτρων που περνάμε σε συναρτήσεις και υποστηρίζονται από τους ANSI C compilers και τους compilers της C++. 'Ενα πρότυπο συνάρτησης έχει τη μορφή:

```
τύπος_δεδομένου όνομα_συνάρτησης(τ1, τ2, ..., τν);
```

όπου ο τύπος_δεδομένου και το όνομα_συνάρτησης έχουν την ίδια σημασία με αυτά που είδαμε στον ορισμό των συναρτήσεων, ενώ τα τ_1, τ_2, ..., τ_ν είναι τύποι δεδομένων και δείχνουν ότι η συνάρτηση δέχεται ν-παραμέτρους, για τις οποίες ισχύει ότι ο τύπος της πρώτης είναι τ_1, ο τύπος της δεύτερης τ_2, κ.ο.κ. Το πρότυπο της συνάρτησης πρέπει να βρίσκεται πριν χρησιμοποιηθεί αυτή για πρώτη φορά. Από τη στιγμή που ο compiler έχει στη διάθεσή του το πρότυπο της συνάρτησης μπορεί να μας ειδοποιήσει αν έχουμε αριθμό παραμέτρων. Αν ο τύπος μιας παραμέτρου που περνάμε δεν συμφωνεί με τον αντίστοιχο τύπο στο πρότυπο συνάρτησης, ο compiler θα προσπαθήσει να κάνει την κατάλληλη μετατροπή και, αν κάτι τέτοιο είναι αδύνατο ή μπορεί να έχει συνέπειες στο πρόγραμμά μας θα μας δώσει κάποιο προειδοποιητικό μήνυμα. Στην κλασική C έπρεπε εμείς να φροντίζουμε για τις μετατροπές αυτές τοποθετώντας δηλώσεις αλλαγής τύπων (type casts) πριν από τις παραμέτρους. Μία δήλωση αλλαγής τύπου έχει τη μορφή
(νέος τύπος)έκφραση
και το αποτέλεσμά της είναι να μετατραπεί ο τύπος της έκφρασης στον νέο τύπο που καθορίσαμε. Αν π.χ. θέλουμε να βρούμε το άθροισμα των ακεραίων μεταβλητών i και j χρησιμοποιώντας τη συνάρτηση sum του σχήματος 3 θα έπρεπε να γράψουμε:

```
sum((double)i, (double)j)
```

Αν παραλείπαμε κάποιο type cast της μορφής αυτής τότε τα αποτελέσματα που θα παίρναμε θα ήταν εσφαλμένα.
Σύμφωνα με τα παραπάνω το πρότυπο συνάρτησης της sum() στο σχήμα 3 θα ήταν:

```
double sum(double, double);
```

Αν η συνάρτηση δεν παίρνει παραμέτρους τότε ο χώρος ανάμεσα στις παρενθέσεις δεν πρέπει να μείνει κενός, αλλά πρέπει να συμπληρωθεί εκεί η λέξη void. Αν δεν βάλουμε τη λέξη void τότε ο compiler δεν θα ελέγχει το πλήθος και τους τύπους των παραμέτρων. Αυτό γίνεται για λόγους συμβατότητας με την κλασική (μη-ANSI) C.
Αντίστοιχα στην ANSI C έχει αλλάξει και ο τρόπος με τον οποίο ορίζουμε τις συναρτήσεις, με το να μετακινηθούν οι δηλώσεις των τύπων των παραμέτρων μαζί με τις δηλώσεις των ονομάτων τους. 'Ετσι μία συνάρτηση στην ANSI C ορίζεται με:

```
τύπος_δεδομένου όνομα(τ1 ο1, τ2 ο2, ..., τν ον)
{
	δηλώσεις δεδομένων;
	εντολές;
}
```

Στην πρώτη γραμμή υπάρχει ο τύπος_δεδομένου και το όνομα, τα οποία δεν έχουν αλλάξει. Αυτό που άλλαξε είναι η μορφή της λίστας των παραμέτρων, για τις οποίες εκτός από το όνομα (o_1) δηλώνουμε στο σημείο εκείνο και τον τύπο της (τ_1). 'Ετσι, η πιο πάνω δήλωση ορίζει μία συνάρτηση με το όνομα *όνομα* που επιστρέφει ένα δεδομένο με τύπο *τύπος_δεδομένου* και δέχεται ν-παραμέτρους. Ο τύπος της πρώτης θα είναι τ_1 και θα μπορούμε μέσα στη συνάρτηση να αναφερόμαστε στην τιμή της με το όνομα o_1, ο τύπος της δεύτερης τ_2 και θα μπορούμε μέσα στη συνάρτηση να αναφερόμαστε στην τιμή της με το όνομα o_2, κ.τ.λ. Φυσικά δεν υπάρχουν πλέον οι δηλώσεις των τύπων των παραμέτρων μετά από την επικεφαλίδα, αφού έχουν μεταφερθεί μέσα στην επικεφαλίδα. Αν μία τέτοια υλοποίηση μιας συνάρτησης βρίσκεται πριν από την πρώτη χρησιμοποίησή της, τότε δεν είναι απαραίτητο να

δώσουμε το πρότυπό της για να έχουμε έλεγχο του πλήθους και του τύπου των παραμέτρων.

5. 'Ελεγχος ροής προγραμμάτων στη C

Ο έλεγχος ροής των προγραμμάτων στη C γίνεται με τις εντολές αποφάσεων, τις εντολές επανάληψης και την εντολή άλματος. Οι εντολές αυτές αναλύονται στις πιο κάτω παραγράφους.

Εντολές αποφάσεων

Οι εντολές αποφάσεων ονομάζονται έτσι γιατί μπορούν να εκτελέσουν ή να μην εκτελέσουν κάποιο κομμάτι κώδικα (κάποιες εντολές δηλαδή) ανάλογα με την τιμή κάποιας έκφρασης. Οι εντολές αποφάσεων είναι οι if/else και switch.
Η σύνταξη της εντολής if είναι:

```
if (συνθήκη)
      εντολή1
[else
      εντολή2]
```

(οι τετράγωνες αγκύλες σημαίνουν ότι το περιεχόμενό τους είναι προαιρετικό). 'Οταν εκτελείται η εντολή if ο compiler εξετάζει την τιμή της συνθήκης. Αν η τιμή αντιστοιχεί στο αληθές (είναι δηλαδή μη μηδενική) τότε θα εκτελεστεί η εντολή$_1$, ενώ η εντολή εντολή$_2$, αν υπάρχει, αγνοείται. Αν η τιμή της συνθήκης αντιστοιχεί στο λογικό ψευδές, τότε η εντολή$_1$ αγνοείται, ενώ αν υπάρχει το else, εκτελείται η εντολή$_2$. Αν το else δεν υπάρχει, τότε δεν γίνεται τίποτε.
Στη θέση των εντολών εντολή$_1$ ή και εντολή$_2$ μπορούμε να έχουμε μόνο μία εντολή ή μία σύνθετη εντολή. Ως σύνθετη εντολή ορίζεται μία λίστα εντολών που περικλείεται ανάμεσα σε μία αριστερή αγκύλη ({) και σε μία δεξιά αγκύλη (}) και μπορεί να περιέχει και δηλώσεις δεδομένων, οι οποίες πρέπει να βρίσκονται αμέσως μετά την αριστερή αγκύλη και πριν την πρώτη εκτελέσιμη εντολή της σύνθετης εντολής και μπορούν να έχουν αρχικοποιήσεις. Οι μεταβλητές αυτές δεν έχουν καμία σχέση με άλλες μεταβλητές με το ίδιο όνομα που εμφανίζονται σε άλλα σημεία του προγράμματος, μπορούν δε να χρησιμοποιηθούν μόνο μέσα στη σύνθετη εντολή. Το αποτέλεσμα της εκτέλεσης μιας σύνθετης εντολής είναι η εκτέλεση των εντολών που περικλείει, αρχίζοντας από την πρώτη και φτάνοντας ως την τελευταία. Τόσο η εντολή$_1$ όσο και η εντολή$_2$ μπορεί με τη σειρά τους να είναι εντολές if/else, θα πρέπει να προσέχουμε όμως, γιατί ο compiler θεωρεί ότι ένα else αντιστοιχεί στο πλησιέστερο if. 'Ετσι η εντολή:

```
if (συνθήκη1)
if (συνθήκη2)
εντολή1
else
εντολή2
```

ερμηνεύεται από τον compiler ως:

α) Αν η συνθήκη$_1$ είναι αληθής, τότε:
   1) αν η συνθήκη2 είναι αληθής, τότε εκτελείται η εντολή$_1$
   2) αν η συνθήκη$_2$ είναι ψευδής, τότε εκτελείται η εντολή$_2$

β) αν η συνθήκη$_1$ είναι ψευδής, δεν γίνεται καμία ενέργεια (αφού δεν υπάρχει else στο if (συνθήκη$_1$)).

Αν θέλουμε να αντιστοιχίσει ο compiler το else στο πρώτο if, τότε θα πρέπει να ορίσουμε το δεύτερο if ως σύνθετη εντολή:

```
if (συνθήκη1)
{
      if (συνθήκη2)
            εντολή1
}
```

```
else
      εντολή2
```

το οποίο ερμηνεύεται από τον compiler ως:

α) αν η $συνθήκη_1$ είναι αληθής, τότε αν η $συνθήκη_2$ είναι αληθής τότε εκτελείται η $εντολή_1$

β) αν η $συνθήκη_1$ είναι ψευδής, τότε εκτελείται η $εντολή_2$

Αν η $συνθήκη_1$ είναι αληθής και η $συνθήκη_2$ ψευδής τότε δεν εκτελείται καμία εντολή, γιατί το δεύτερο if δεν έχει else.

Η συνάρτηση *print_time* του σχήματος 4 εκτυπώνει την ώρα που δίνεται ως δύο παράμετροι (hour και min) σε μορφή ρολογιού 24 ωρών ή σε μορφή ρολογιού 12 ωρών. Η μορφή που θα χρησιμοποιηθεί καθορίζεται από την παράμετρο kind. Η παράμετρος hour αντιπροσωπεύει την ώρα σε μορφή ρολογιού 24 ωρών.

```
void print_time(int hour, int min, int kind)
{
      if (kind)
            /* χρησιμοποίηση ρολογιού 24 ωρών */
            printf("%d:%d\n", hour, min);
      else /* αν το kind είναι 0 */
      /* χρησιμοποίηση ρολογιού 12 ωρών */
            if (hour <= 12)
                  printf("%d:%d am\n", hour, min);
            else
                  printf("%d:%d pm\n", hour - 12, min);
}
```

Σχήμα 4

Η σύνταξη της εντολής switch είναι:

```
switch (έκφραση)
{
      case σταθερά1:
            εντολές1
            break;
      case σταθερά2:
            εντολές2
            break;
...
      case σταθεράν:
            εντολέςν
            break;
[default:
            εντολέςx
            break;]
}
```

Ο τύπος της έκφρασης έκφραση θα πρέπει να είναι κάποιος από αυτούς που υποστηρίζει άμεσα η γλώσσα και οι τύποι των σταθερών $σταθερά_ι$ θα πρέπει να συμφωνούν με αυτόν της έκφρασης. Η εντολή switch λειτουργεί κατά τον ακόλουθο τρόπο: Αρχικά υπολογίζεται η τιμή της έκφρασης και κατόπιν συγκρίνεται με τη $σταθερά_1$. Αν είναι ίσες, τότε εκτελούνται οι $εντολές_1$ (μπορεί να είναι και περισσότερες από μία) και η εντολή switch τερματίζει. Αν η τιμή της έκφρασης δεν είναι ίση με τη $σταθερά_1$, τότε συγκρίνεται με τη $σταθερά_2$, και αν είναι ίσες εκτελούνται οι $εντολές_2$ και η switch τερματίζει. Η διαδικασία αυτή επαναλαμβάνεται για όλα τα "case $σταθερά_ι$:" της switch, και αν δεν βρεθεί ταίριασμα τότε:

α) αν υπάρχει το τμήμα *default:* τότε εκτελούνται οι $εντολές_x$

β) αν δεν υπάρχει το τμήμα *default:* δεν γίνεται τίποτε.

Η ύπαρξη του break; μετά τις εντολές $εντολές_1$ δεν είναι απαραίτητη συντακτικά, αλλά αν δεν υπάρχει και η τιμή της έκφρασης είναι ίση με τη $σταθερά_1$, τότε θα εκτελεστούν όλες οι εντολές από το σημείο "case $σταθερά_1$:" μέχρι το τέλος τη switch ή μέχρι το πρώτο break.
Η συνάρτηση print_month του σχήματος 5 τυπώνει το πλήρες όνομα ενός μήνα, ο οποίος καθορίζεται από την παράμετρο month. Αν η παράμετρος δεν βρίσκεται στο διάστημα 1-12 εκτυπώνεται ένα μήνυμα λάθους.

```
void print_month(int month)
{
	switch (month)
	{
		case 1:
			printf("January\n"); break;
		case 2:
			printf("February\n"); break;
		case 3:
			printf("March\n"); break;
		case 4:
			printf("April\n"); break;
		case 5:
			printf("May\n"); break;
		case 6:
			printf("June\n"); break;
		case 7:
			printf("July\n"); break;
		case 8:
			printf("August\n"); break;
		case 9:
			printf("September\n"); break;
		case 10:
			printf("October\n"); break;
		case 11:
			printf("November\n"); break;
		case 12:
			printf("December\n"); break;
		default:
			printf("Illegal month name.\n"); break;
	}
}
```

Σχήμα 5

Οι εντολές επανάληψης
Η C παρέχει τρεις εντολές επανάληψης, τη while, τη do-while και τη for.
Η σύνταξη της εντολής while είναι η ακόλουθη:

```
while (συνθήκη)
	εντολή
```

Η εντολή *εντολή* πρέπει να είναι μία αλλά μπορούμε να χρησιμοποιήσουμε σύνθετη εντολή, και η εκτέλεσή της γίνεται ως εξής: Αρχικά αποτιμάται η τιμή της συνθήκης *συνθήκη*. Αν αντιστοιχεί στο λογικό αληθές, τότε εκτελείται η εντολή *εντολή* και η διαδικασία επαναλαμβάνεται. Αν η τιμή της συνθήκης αντιστοιχεί στο λογικό ψευδές τότε η εντολή *εντολή* δεν εκτελείται, και η εντολή while τερματίζεται. Το πρόγραμμα του σχήματος 6 τυπώνει τους αριθμούς από το 1 έως το 10 και τα διπλάσιά τους.

```
#include <stdio.h>

main()
```

```
{
	int i = 10, i2;

	while (i)
	{
		i2 = 2 * i;
		printf("%d\t%d\n", i, i2);
		i = i - 1;
	}
	return 0;
}
```

Σχήμα 6

Η εντολή "i = i - 1;" του πιο πάνω σχήματος ερμηνεύεται ως "θέσε ως νέα τιμή της μεταβλητής i την τρέχουσα τιμή του i ελαττωμένη κατά 1" και δεν αντιστοιχεί σε κάποια εξίσωση, η οποία θα ήταν ούτως ή άλλως αδύνατη.
Η σύνταξη της εντολής do-while είναι η εξής:

```
do
	εντολή
while (συνθήκη);
```

Η διαδικασία εκτέλεσης μιας εντολής do-while είναι η ακόλουθη: πρώτα εκτελείται η εντολή *εντολή* και στη συνέχεια υπολογίζεται η τιμή της συνθήκης *συνθήκη*. Αν η τιμή αντιστοιχεί στο λογικό αληθές, τότε η διαδικασία επαναλαμβάνεται, ενώ αν αντιστοιχεί στο λογικό ψευδές η εντολή τερματίζεται. Γενικά η εντολή do-while είναι ισοδύναμη με τη while· η διαφορά τους έγκειται στο ότι η εντολή *εντολή* στο do-while θα εκτελεστεί τουλάχιστον μία φορά, ενώ στο while μπορεί να μην εκτελεστεί καθόλου.
Η εντολή for έχει λειτουργικότητα ανάλογη με αυτή της while αλλά μας βοηθάει να κρατάμε τις εντολές που καθορίζουν την πορεία εκτέλεσης της επαναληπτικής διαδικασίας συγκεντρωμένες. Η σύνταξή της είναι η ακόλουθη:

```
for (εντολές-for-1; συνθήκη; εντολές-for-2)
	εντολή
```

Κάθε ένα από τα τμήματα εντολές-for-1, συνθήκη και εντολές-for-2 μπορεί να παραλειφθεί. Οι εντολές-for-1 και εντολές-for-2, αν δεν παραλειφθούν, είναι σειρές από εκχωρήσεις (εντολές της μορφής α = β) και κλήσεις σε συναρτήσεις, χωρισμένες με κόμμα (στη γενική περίπτωση είναι εκφράσεις, οι οποίες θα αναλυθούν στο επόμενο κεφάλαιο). Η εκτέλεση εντολών χωρισμένων με κόμμα γίνεται από την πρώτη προς την τελευταία. Η εντολή for εκτελείται ως εξής:

1) εκτελούνται οι εντολές *εντολές-for-1*
2) υπολογίζεται η τιμή της συνθήκης *συνθήκη*. Αν η τιμή αντιστοιχεί στο λογικό ψευδές η εντολή for τερματίζεται (και τα βήματα 3 και 4 δεν εκτελούνται). Αν η συνθήκη έχει παραλειφθεί τότε η τιμή της θεωρείται ότι είναι πάντα το λογικό αληθές
3) εκτελείται η εντολή *εντολή*
4) εκτελούνται οι εντολές *εντολές-for-2*
5) η διαδικασία επαναλαμβάνεται από το βήμα 2

Σύμφωνα με τα πιο πάνω, η εντολή for μπορεί να γραφεί με τη χρήση της εντολής while όπως φαίνεται στο σχήμα 7:

```
εντολές-for-1;
while (συνθήκη)
{
	εντολή
```

```
        εντολές-for-2;
}
```

Σχήμα 7

Σε όλες τις εντολές επανάληψης μπορούν να χρησιμοποιηθούν μέσα στην εντολή *εντολή* οι εντολές *break* και *continue* (αυτό συνήθως γίνεται όταν αυτή είναι σύνθετη). 'Οταν εκτελεστεί μία εντολή break μέσα σε μία εντολή επανάληψης τότε η εντολή αυτή τερματίζεται, ενώ όταν εκτελεστεί μια εντολή continue τότε αγνοούνται οι υπόλοιπες εντολές της -σύνθετης- εντολής εντολή και αρχίζει η επόμενη επανάληψη. 'Ετσι το πρόγραμμα του σχήματος 8 θα τυπώσει τους αριθμούς από το 10 μέχρι το 6, αν και η συνθήκη του for θα εξακολουθεί να είναι αληθής, ενώ το πρόγραμμα του σχήματος 9 θα τυπώσει τους αριθμούς από το 10 μέχρι το 1 εκτός από το 5.

```
#include <stdio.h>

main()
{
        int i;
        for (i = 10; i; i = i - 1)
        {
                if (i == 5)
                        break;
                printf("%d\n", i);
        }
        return 0;
}
```

Σχήμα 8

```
#include <stdio.h>

main()
{
        int i;
        for (i = 10; i; i = i - 1)
        {
                if (i == 5)
                        continue;
                printf("%d\n", i);
        }
        return 0;
}
```

Σχήμα 9

Ο συμβολισμός "a == b" στη C είναι ο έλεγχος ισότητας. Το αποτέλεσμά του είναι μη-μηδενικό αν οι τιμές των a και b είναι ίσες και μηδενικό στην αντίθετη περίπτωση.

Η εντολή άλματος

Με την εντολή goto στη C μπορούμε να μεταφέρουμε τον έλεγχο του προγράμματος από ένα σημείο μιας συνάρτησης σε ένα άλλο. Η σύνταξή της είναι:

```
goto label;
```

ενώ σε κάποιο σημείο της ίδιας συνάρτησης θα πρέπει να υπάρχει η δήλωση

```
label:
```

'Οταν εκτελείται η εντολή goto το πρόγραμμά μας συνεχίζει να εκτελείται από το σημείο που βρίσκεται η δήλωση label: και μετά. Η εντολή goto είναι καλό να μη χρησιμοποιείται γιατί περιπλέκει τα προγράμματά μας και είναι δύσκολο να τα κατανοήσουμε στη συνέχεια. Η εντολή goto μπορεί πάντα να αποφευχθεί με χρήση ίσως μίας επιπλέον μεταβλητής και των εντολών επανάληψης και αποφάσεων. Η μόνη ίσως περίπτωση στην οποία θα πρέπει να χρησιμοποιείται το goto είναι όταν θέλουμε να βγούμε από μια σειρά επαναληπτικών εντολών που η μία εμπεριέχει την άλλη:

```
for (...)
      for (...)
      ...
                        for (...)
                        {
                              if (disaster)
                                    goto error:
                        }
...
error:
```

Αν η εντολή goto οδηγεί μέσα σε μία σύνθετη εντολή που περιέχει δηλώσεις μεταβλητών με αρχικοποιήσεις, τότε οι αρχικοποιήσεις αυτές δεν θα γίνουν. Δεν μπορούμε να χρησιμοποιήσουμε την εντολή goto για να πάμε από μία συνάρτηση σε άλλη.

6. Εκφράσεις και τελεστές στη C

Σε ένα πρόγραμμα C τα πάντα (εκτός από τις κλήσεις σε συναρτήσεις με τύπο επιστροφής void) θεωρούνται εκφράσεις, έχουν δηλαδή κάποια τιμή που μπορούμε να χρησιμοποιήσουμε ή να αγνοήσουμε. Μία έκφραση μπορεί να είναι μία σταθερά, μία μεταβλητή ή μία κλήση σε συνάρτηση με τύπο επιστροφής διαφορετικό void. Δύο ή περισσότερες εκφράσεις μπορούν να συνδυαστούν σε μία πιο πολύπλοκη έκφραση με τη βοήθεια των τελεστών. Οι τελεστές που είναι διαθέσιμοι στη C φαίνονται στον πιο κάτω πίνακα (οι τελεστές δίνονται κατά φθίνουσα σειρά προτεραιότητας).

```
()  []  ->  .
!  ~  ++  --  +  -  *  &  (τύπος)  sizeof
*  /  %
+  -
<<  >>
<  <=  >  >=
==  !=
&
^
|
&&
||
?:
=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=
, (κόμμα)
```

Πίνακας 1

Οι τελεστές +, - * και & εμφανίζονται δύο φορές. Η πρώτη εμφάνισή τους αντιστοιχεί σε μοναδιαία χρήση (δηλαδή ο τελεστής εφαρμόζεται σε ένα μόνο όρισμα) ενώ ο η δεύτερη εμφάνισή τους αντιστοιχεί σε δυαδική χρήση (με άλλα λόγια ο τελεστής χρειάζεται δύο ορίσματα). Οι τελεστές *[]*, ->, ., * (μοναδιαίο) και & (μοναδιαίο) θα αναπτυχθούν σε επόμενα κεφάλαια.

Οι τελεστές + και - στη μοναδιαία τους μορφή αντιστοιχούν σε προσήμανση σταθερών, μεταβλητών ή εκφράσεων. Οι τελεστές +, -, * (στη δυαδική τους μορφή) και / αντιστοιχούν στις γνωστές πράξεις της αριθμητικής, δηλαδή πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση. Αν και τα δύο ορίσματα του τελεστή είναι του ίδιου τύπου τότε το αποτέλεσμα είναι του ίδιου τύπου με τα ορίσματα. Αν οι τύποι των ορισμάτων είναι διαφορετικοί, τότε ο τύπος του αποτελέσματος είναι ο ισχυρότερος από τους δύο τύπους. Οι τύποι δίνονται στον πιο κάτω πίνακα κατά φθίνουσα σειρά ισχύος:

```
long double
double
float
unsigned   long
int
long int
unsigned int
int
```

Πίνακας 2

Οι τύποι unsigned short int, short int και char δεν εμφανίζονται γιατί αν πρόκειται να γίνουν πράξεις σε δεδομένα τέτοιου τύπου, τα δεδομένα μετατρέπονται σε τύπου int,

γίνεται η πράξη και το αποτέλεσμα μετατρέπεται ξανά στον κατάλληλο τύπο, αν κάτι τέτοιο είναι απαραίτητο.
Ιδιαίτερη προσοχή χρειάζεται η χρήση του τελεστή / (διαίρεση) στους ακεραίους. 'Οπως είπαμε το αποτέλεσμα της είναι ακέραιος, γεγονός που μεταφράζεται σε απώλεια (όχι στρογγύλευση) των δεκαδικών ψηφίων. 'Ετσι το αποτέλεσμα της έκφρασης
9 / 4 * 5
είναι 10 (γιατί αποτιμάται ως (9 / 4) * 5 = 2 * 5 = 10), που είναι διαφορετικό από το αποτέλεσμα της έκφρασης
9 * 5 / 4
το οποίο είναι 11 (γιατί αποτιμάται ως (9 * 5) / 4 = 45 / 4 = 11). Στη γενική περίπτωση, όταν έχουμε πολλαπλασιασμούς και διαιρέσεις ακεραίων θα πάρουμε πιο ακριβή αποτελέσματα αν κάνουμε πρώτα όλους τους πολλαπλασιασμούς και στη συνέχεια τις διαιρέσεις. Αν θέλουμε να πάρουμε και τα δεκαδικά ψηφία της διαίρεσης θα πρέπει να μετατρέψουμε το ένα από τα δύο ορίσματα της διαίρεσης σε κάποιο τύπο που αναπαριστά πραγματικούς (float, double, long double) με τη χρήση κάποιου type cast. Π.χ. αν η μεταβλητές i και j είναι ακέραιες και θέλουμε να πάρουμε και τα δεκαδικά ψηφία της διαίρεσης i / j θα πρέπει να το συμβολίσουμε ως

```
(double)i / j
```

(ή i / (double) j). Δεν είναι απαραίτητο να αλλάξουμε τον τύπο του άλλου ορίσματος γιατί αυτό γίνεται αυτόματα.
Ο τελεστής *%* εφαρμόζεται μόνο σε τύπους δεδομένων που αναπαριστούν ακεραίους και επιστρέφει το υπόλοιπο της διαίρεσης μεταξύ των ορισμάτων του. Αν κάποιο από τα ορίσματά του είναι αρνητικός τότε ενδέχεται τα αποτελέσματα να μην είναι σωστά.
Οι τελεστές *<*, *<=*, *>*, *>=*, *==* και *!=* αντιπροσωπεύουν σύγκριση μεταξύ των ορισμάτων τους (μικρότερο, μικρότερο ή ίσο, μεγαλύτερο, μεγαλύτερο ή ίσο, ίσο και διάφορο αντίστοιχα). Το αποτέλεσμά τους είναι πάντα ακέραιος και είναι μηδέν αν δεν ισχύει η σύγκριση και διαφορετικό από μηδέν αν ισχύει.
Ο τελεστής *?:* είναι ο μοναδικός τριαδικός τελεστής που υπάρχει στη C. Η σύνταξή του είναι:
(συνθήκη) ? έκφραση$_1$: έκφραση$_2$
και έχει την εξής σημασία: Αν η συνθήκη είναι αληθής, τότε το αποτέλεσμα του τελεστή είναι η έκφραση$_1$, αν όχι, τότε το αποτέλεσμα του τελεστή είναι η έκφραση$_2$. Ο τύπος του αποτελέσματος είναι ο ισχυρότερος από τους τύπους της έκφρασης$_1$ και της έκφρασης$_2$. 'Ετσι η έκφραση

```
(a > b) ? a : b
```

έχει ως αποτέλεσμα τον μεγαλύτερο από τους a και b. Οι παρενθέσεις γύρω από το a > b δεν είναι απαραίτητες γιατί ο τελεστής > έχει μεγαλύτερη προτεραιότητα από τον ?:, αλλά χρησιμοποιούνται για λόγους σαφήνειας. Παρενθέσεις πρέπει να χρησιμοποιήσουμε όπου επιθυμούμε διαφορετική σειρά εφαρμογής των τελεστών από την προκαθορισμένη. 'Ετσι για να προσθέσουμε 1 στη μεταβλητή a και να πολλαπλασιάσουμε το αποτέλεσμα της πρόσθεσης επί δύο θα πρέπει να γράψουμε

```
(a + 1) * 2
```

γιατί η γραφή

```
a + 1 * 2
```

θα έχει ως αποτέλεσμα να γίνει πρώτα ο πολλαπλασιασμός 1 * 2 και κατόπιν να προστεθεί το αποτέλεσμά του (2) στο a. Το τελικό αποτέλεσμα της έκφρασης είναι το a + 2.
Ο τελεστής *!* αντιπροσωπεύει το λογικό όχι. Παίρνει ένα όρισμα και επιστρέφει έναν ακέραιο που αντιστοιχεί στο λογικό αληθές (κάτι μη μηδενικό) όταν το όρισμά του

είναι ψευδές (μηδέν) και ψευδές όταν το όρισμά του είναι αληθές. Οι τελεστές *&&* και || αντιστοιχούν στη λογική σύζευξη και στη λογική διάζευξη αντίστοιχα. Και οι δύο είναι δυαδικοί, παίρνουν δηλαδή δύο ορίσματα. Ο τελεστής *&&* επιστρέφει αληθές αν και τα δύο ορίσματά του είναι αληθή και ψευδές σε κάθε άλλη περίπτωση. Αν το αποτέλεσμα του πρώτου ορίσματος είναι ψευδές, τότε δεν ελέγχεται η τιμή του δευτέρου ορίσματος. Ο τελεστής || επιστρέφει αληθές όταν τουλάχιστον ένα από τα ορίσματά του είναι αληθές και ψευδές όταν και τα δύο είναι ψευδή. Αν το πρώτο όρισμα είναι αληθές, η τιμή του δεύτερου δεν ελέγχεται. 'Ετσι η έκφραση

```
a > 5 && a < 10
```

είναι αληθής αν το a έχει τιμή μεγαλύτερη του 5 και μικρότερη του 10, ενώ η έκφραση

```
a <= 5 || a >= 10
```

είναι ακριβώς το αντίθετο της προηγούμενης, είναι δηλαδή αληθής όταν η τιμή του a είναι μικρότερη ή ίση του 5 ή μεγαλύτερη ή ίση του 10.

Ο τελεστής *sizeof* έχει δύο δυνατές συντάξεις:

α) sizeof μεταβλητή

β) sizeof(τύπος δεδομένου)

Και στις δύο περιπτώσεις επιστρέφει το πλήθος των bytes που απαιτούνται για την αποθήκευση της μεταβλητής μεταβλητή ή μιας μεταβλητής τύπου τύπος δεδομένου αντίστοιχα. Η χρήση του τελεστή sizeof θα αναπτυχθεί σε επόμενο κεφάλαιο.

Οι τελεστές <<, >>, ~, *&* (δυαδικό), ^ και | χρησιμοποιούνται για το χειρισμό ακεραίων ποσοτήτων (char, short, int, long και τα αντίστοιχα unsigned) σε επίπεδο bit. 'Ολοι είναι δυαδικοί, παίρνουν δηλαδή δύο ορίσματα τα οποία πρέπει να είναι ακέραιοι.

Ο τελεστής ~ αντιπροσωπεύει το bitwise not. Το αποτέλεσμά του είναι ένας ακέραιος του οποίου το ι-οστό bit είναι 1 αν το ι-οστό bit του ορίσματός του είναι 0, και 0 αν το ι-οστό bit του ορίσματός του είναι 1. 'Ετσι η έκφραση

```
~18020
```

δίνει ως αποτέλεσμα:

```
~18020 = ~0100011001100100 = 1011100110011011 =47515
```

Ο τελεστής *&* αντιπροσωπεύει το bitwise and, το αποτέλεσμά του δηλαδή είναι ένας ακέραιος του οποίου το ι-οστό bit προκύπτει από το and μεταξύ του ι-οστού bit του πρώτου ορίσματος και του ι-οστού bit του δεύτερου ορίσματος. Το αποτέλεσμα του and μεταξύ δύο bits είναι 1 αν και τα δύο bits είναι 1 και 0 στην αντίθετη περίπτωση. Αν έχουμε έτσι την έκφραση

```
18020 & 27321
```

θα πάρουμε ως αποτέλεσμα:

```
18020 = 0100011001100100
27321 = 0110101010111001 &
__________________________
        0100001000100000 = 16928
```

Ο τελεστής ^ αντιπροσωπεύει το bitwise xor, το αποτέλεσμά του είναι ένας ακέραιος του οποίου το ι-οστό bit είναι το xor μεταξύ του ι-οστού bit του πρώτου ορίσματος και του ι-οστού bit του δεύτερου ορίσματος. Το αποτέλεσμα του xor μεταξύ δύο bits είναι 1 αν τα δύο bits είναι διαφορετικά και 0 αν είναι ίδια. 'Ετσι η έκφραση

```
18020 ^ 27321
```

δίνει ως αποτέλεσμα:

```
18020 = 0100011001100100
27321 = 0110101010111001 ^
__________________________
        0010110011011101   = 11449
```

Ο τελεστής | αντιπροσωπεύει το bitwise or, το αποτέλεσμά του δηλαδή είναι ένας ακέραιος του οποίου το ι-οστό bit είναι το or μεταξύ του ι-οστού bit του πρώτου ορίσματος και του ι-οστού bit του δεύτερου ορίσματος. Το αποτέλεσμα του xor

μεταξύ δύο bits είναι 1 αν κάποιο από τα δύο bits είναι 1 και 0 αν και τα δύο bits είναι 0. Έτσι η έχουμε έτσι την έκφραση

```
18020 | 27321
```

δίνει ως αποτέλεσμα:

```
18020 = 0100011001100100
27321 = 0110101010111001 |
        0110111011111101   = 28413
```

Οι τελεστές << και >> μετακινούν τα bits ενός ακεραίου αριστερά και δεξιά αντίστοιχα, γεμίζοντας τις κενές -πλέον- θέσεις με μηδενικά και απορρίπτοντας τα bits που δεν χωράνε πλέον στον ακέραιο. Και οι δύο παίρνουν δύο ορίσματα, από τα οποία το αριστερό όρισμα ορίζει τον ακέραιο του οποίου τα bits θα μετακινηθούν και το δεξιό όρισμα το πόσες θέσεις θα μετακινηθούν, αριστερά για τον τελεστή <<, δεξιά για τον τελεστή >>. Έτσι έχουμε ότι:

```
18020 <<3 = 0100011001100100 <<3 = 0011001100100000 = 13088
18020 >>3 = 0100011001100100 >>3 = 0000100011001100 = 2252
```

Ο τύπος του αποτελέσματος των τελεστών << και >> είναι ο ίδιος με το αριστερό τους όρισμα.

Οι τελεστές που χειρίζονται ακεραίους σε επίπεδο bit μπορούν να χρησιμοποιηθούν, μεταξύ άλλων, για να τοποθετηθούν σε μία ακέραιη ποσότητα πληροφορίες που αν αποθηκευόταν ξεχωριστά θα χρειαζόταν περισσότερο χώρο αποθήκευσης. Π.χ. η αποθήκευση της ημερομηνίας που θα χρειαζόταν τρεις ακεραίους (ημέρα, μήνας και έτος) αποθηκεύεται από το MS-DOS σε έναν ακέραιο 16 bits κατά τον ακόλουθο τρόπο: Τα 7 υψηλότερης τάξης bits του ακεραίου αναπαριστούν τα έτη που πρέπει να προστεθούν στο 1980 για να έχουμε το σωστό έτος, τα επόμενα 4 αναπαριστούν το μήνα (0-11) και τα τελευταία 5 την ημέρα (0-31). Στο σχήμα 10 βλέπουμε δύο συναρτήσεις, την print_date() και την make_date(). Η print_date() δέχεται ως παράμετρο έναν ακέραιο που αντιπροσωπεύει μία ημερομηνία του MS-DOS και τυπώνει την ημερομηνία που παριστάνει στη μορφή ηη-μμ-εεεε, ενώ η make_date() δέχεται ως παραμέτρους τρεις ακεραίους που αναπαριστούν ημέρα, μήνα και έτος μιας ημερομηνίας και επιστρέφει έναν ακέραιο που αντιστοιχεί στην MS-DOS αναπαράσταση της ημερομηνίας αυτής. Λόγω του ότι το MS-DOS χρησιμοποιεί 7 bits για το έτος, θα πρέπει το έτος στη make_date() να είναι μεγαλύτερο ή ίσο του 1980 και μικρότερο ή ίσο του 2027.

```
void print_date(int ms_dos_date)
{
	int day, month, year;

	day = ms_dos_date & 0x1f;
	month = (ms_dos_date >> 5) & 0xf;
	year = ms_dos_date >> 9;
	printf("%d-%d-%d\n", day, month, year);
}
int make_date(int day, int month, int year)
{
	int ms_dos_date;

	if ((year < 1980) || (year > 2127))
		return 0;
	ms_dos_date = day;
	ms_dos_date = ms_dos_date | (month << 5);
	ms_dos_date = ms_dos_date | ((year - 1980) << 9);
	return ms_dos_date;
}
```

Σχήμα 10

Ο τελεστής = ορίζει την εκχώρηση τιμής όπως έχουμε ήδη δει. Στο αριστερό του μέρος πρέπει να βρίσκεται μία μεταβλητή και στο δεξιό οποιαδήποτε έκφραση. Στη C η εκχώρηση είναι έκφραση, και το αποτέλεσμά της είναι η τιμή που εκχωρείται στη μεταβλητή που βρίσκεται στο αριστερό μέρος του τελεστή. Μπορούμε έτσι να γράψουμε:

```
a = b = 0;
```

που έχει ως αποτέλεσμα να μηδενιστούν οι μεταβλητές a και b. Μπορούμε ακόμη να χρησιμοποιήσουμε την έκφραση:

```
a = (b = c) + 2;
```

που δίνει στο b την τρέχουσα τιμή της μεταβλητής c και στο a την τιμή αυτή αυξημένη κατά 2. Τέτοιες εκφράσεις είναι καλό όμως να αποφεύγονται γιατί είναι εύκολο να κάνουμε λάθος και είναι δύσκολο να κατανοηθούν.

Το γεγονός ότι η εκχώρηση είναι έκφραση δημιουργεί και έναν επιπλέον κίνδυνο: Αν χρησιμοποιηθεί κατά λάθος το = στη θέση του == για τον έλεγχο κάποιας ισότητας (σε εντολή if, for, while, κ.ο.κ.) τότε αυτό δεν είναι συντακτικό λάθος, αλλά μεταφράζεται κανονικά, με τη διαφορά ότι δεν κάνει αυτό που θέλουμε. 'Ετσι η εντολή:

```
if (a = 0)
      printf("a is zero\n");
```

ΔΕΝ ελέγχει αν η τιμή του a είναι 0, αλλά εκχωρεί σ' αυτό την τιμή 0. Η τιμή της έκφρασης *a = 0* είναι πάντα 0, και έτσι η κλήση στη συνάρτηση printf() δεν θα εκτελεστεί ποτέ.

Οι τελεστές +=, -=, *=, /=, %=, &=, ^=, |=, <<= και >>= είναι στην πραγματικότητα συντομογραφίες. Ο συμβολισμός

```
a o= b;
```

(όπου *o* είναι ένα από τα +, -, *, /, %, &, ^, |, << ή >>) είναι ισοδύναμος με το συμβολισμό

```
a = a o b;
```

Οι συντομογραφίες αυτές είναι καλό να χρησιμοποιούνται γιατί είναι πιο απλές στην πληκτρολόγηση, επιτρέπουν στον compiler να παράγει καλύτερο κώδικα και είναι πιο αναγνώσιμες από τις ισοδύναμες πλήρεις εκφράσεις τους. Δύο ακόμη συντομογραφίες είναι οι τελεστές ++ και --, που είναι μοναδιαίοι, το όρισμά τους θα πρέπει να είναι ακέραια μεταβλητή και μπορούν να τοποθετούνται πριν ή μετά από τη μεταβλητή. Ο τελεστής ++ αυξάνει την τιμή της μεταβλητής κατά 1, ενώ ο τελεστής --μειώνει την τιμή της μεταβλητής κατά 1. Το αν η θέση του τελεστή είναι πριν ή μετά από τη μεταβλητή επηρεάζει το αποτέλεσμα που ο τελεστής επιστρέφει: Αν ο τελεστής βρίσκεται μετά από τη μεταβλητή, τότε επιστρέφεται η αρχική τιμή της μεταβλητής (πριν από την αύξηση ή τη μείωση), ενώ αν βρίσκεται πριν από τη μεταβλητή, τότε επιστρέφεται η τελική τιμή της μεταβλητής (μετά την αύξηση ή τη μείωση). Στο σχήμα 11 φαίνεται ένα πρόγραμμα που δείχνει τη χρήση και τη λειτουργία των τελεστών αυτών, καθώς και η έξοδος του προγράμματος.

```
#include <stdio.h>

int main()
{
      int i;

      i = 10;
      printf("printf #1: %d\n", i++);
      printf("printf #2: %d\n", i);
      i = 10;
```

```
        printf("printf #3: %d\n", ++i);
        printf("printf #4: %d\n", i);
        i = 10;
        printf("printf #5: %d\n", i--);
        printf("printf #6: %d\n", i);
        i = 10;
        printf("printf #7: %d\n", --i);
        printf("printf #8: %d\n", i);
        return 0;
}

'Εξοδος του προγράμματος:
printf#1: 10
printf#2: 11
printf#3: 11
printf#4: 11
printf#5: 10
printf#6: 9
printf#7: 9
printf#8: 9
```

Σχήμα 11

Ο τελεστής , (κόμμα) τέλος, συνδέει δύο εκφράσεις. Το αποτέλεσμά του είναι να αποτιμάται η αριστερή έκφραση, η προκύπτουσα τιμή να μη λαμβάνεται υπ' όψιν και το τελικό αποτέλεσμα του τελεστή να είναι η τιμή της δεξιάς έκφρασης. Η μόνη πραγματική χρησιμότητα του τελεστή , είναι στην εντολή for όταν πρέπει να αρχικοποιήσουμε περισσότερες από μία μεταβλητές ή να εκτελούμε περισσότερες από μία εντολές-for πριν επανεξεταστεί η συνθήκη:

```
for (i = 0, j = k + 5; i < 10; i++, j *= 2)
        printf("%d\t%d\n", i, j);
```

7. Arrays και strings

'Ενα array στη C είναι ένας αριθμός από ομοειδή δεδομένα στο καθένα από τα οποία μπορούμε να αναφερθούμε με το όνομα του array και τον αύξοντα αριθμό του μέσα στο array. Για να δηλώσουμε ένα array χρησιμοποιούμε μία δήλωση της μορφής:

```
τύπος_δεδομένου όνομα[πλήθος_στοιχείων];
```

Η πιο πάνω δήλωση ορίζει το array *όνομα* το οποίο αποτελείται από *πλήθος_στοιχείων* δεδομένα, κάθε ένα από τα οποία είναι τύπου *τύπος_δεδομένου*. Το *πλήθος_στοιχείων* θα πρέπει να είναι ακέραια σταθερά (μεγαλύτερη του μηδενός) ή σταθερή έκφραση. Στο πρώτο από τα στοιχεία αυτά μπορούμε να αναφερθούμε με το συμβολισμό *όνομα[0]*, στο δεύτερο με το συμβολισμό *όνομα[1]* και στο τελευταίο με το συμβολισμό *όνομα[πλήθος_στοιχείων - 1]*. Με τον όρο **αναφορά** εννοούμε ότι μπορούμε να χρησιμοποιήσουμε την τιμή του στοιχείου σε μία έκφραση (π.χ. *όνομα[2] + 6*) ή να αλλάξουμε την τιμή του μέσω μιας έκφρασης-εκχώρησης (π.χ. *όνομα[2] = 14;*).
'Ενα array μπορεί, όπως και μία μεταβλητή, να αρχικοποιείται, μόνο όμως όταν η δήλωσή του βρίσκεται έξω από κάθε συνάρτηση, δεν μπορούμε δηλαδή να αρχικοποιήσουμε ένα array το οποίο δηλώνεται ως τοπικό δεδομένο σε μία συνάρτηση ή σε μία σύνθετη εντολή. Η δήλωση ενός array του οποίου τα στοιχεία αρχικοποιούνται έχει τη μορφή:

```
τύπος_δεδομένου όνομα[πλήθος] = {τ1, τ2, ..., τν};
```

Τα $τ_ι$ είναι σταθερές των οποίων ο τύπος συμφωνεί με τον *τύπο_δεδομένου* και το πλήθος τους θα πρέπει να είναι μικρότερο ή ίσο από το *πλήθος* που καθορίζει το μέγεθος του array. Το $τ_1$ θα είναι η αρχική τιμή του στοιχείου *όνομα[0]*, το $τ_2$ η αρχική τιμή του *όνομα[1]*, κ.τ.λ. Αν το ν είναι μικρότερο από το *πλήθος* τότε αρχικοποιούνται μόνο τα ν-πρώτα στοιχεία του array. Αν το ν είναι μεγαλύτερο, τότε έχουμε λάθος.
Σε μία αρχικοποιημένη δήλωση array μπορούμε να παραλείψουμε το πλήθος των στοιχείων που το array θα περιέχει και ο compiler θα το συμπεράνει από το πλήθος των αρχικών τιμών. 'Ετσι η δήλωση
int a[] = {3, 5, 7};
ορίζει το array a με τρία ακέραια στοιχεία, τα οποία θα έχουν αρχικές τιμές 3, 5 και 7.
Η συνάρτηση print_day() του σχήματος 12 παίρνει ως παράμετρο έναν ακέραιο από 1 έως 365 που αντιπροσωπεύει μία ημέρα μέσα στο έτος (τα δίσεκτα έτη δεν καλύπτονται) και τυπώνει τον μήνα και την ημέρα του μήνα που αντιστοιχούν στην ημέρα αυτή.

```
int month_days[12] = {31, 28, 31, 30, 31, 30, 31,
                        30, 31, 30, 31};
void print_day(int day)
{
      int i, month;

      if ((day <= 0) || (day > 365))
      {
            printf("Illegal day.\n");
            return;
      }
      i = day;
      month = 0;
      while (i > month_days[month])
            i -= month_days[month++];
      printf("%d/%d\n", i, month + 1);
```

```
}
```

Σχήμα 12

Μπορούμε να περάσουμε ένα array ως παράμετρο σε μία συνάρτηση, αλλά δεν επιτρέπεται να ορίσουμε κάποια συνάρτηση που να επιστρέφει οποιοδήποτε array. Κατά το πέρασμα μάλιστα ενός array ως παραμέτρου, επιτρέπεται να μη δηλώσουμε το πλήθος στοιχείων του, έτσι ώστε η συνάρτηση να μπορεί να χειριστεί arrays οποιουδήποτε μήκους. Φυσικά το πρόγραμμα θα πρέπει να έχει κάποιο τρόπο να βρίσκει το μήκος του array, πράγμα που συνήθως επιτυγχάνεται είτε με μία παράμετρο, είτε με το να παίρνει το τελευταίο στοιχείο του array κάποια ειδική τιμή. Στο πρόγραμμα του σχήματος 13 ορίζεται και χρησιμοποιείται μία τέτοια συνάρτηση η print_array_of_int().

```
#include <stdio.h>

void print_array_of_int(int array[], int num_elements)
{
	int i;

	for (i = 0; i < num_elements; i++)
		printf("%d, ", array[i]);
	printf("\n");
}

int main()
{
	int array[2], array2[3];

	array[0] = 7; array[1] = 20;
	array2[0] = 14; array2[1] = array[0]; array2[2] = 31;
	print_array_of_int(array, 2);
	print_array_of_int(array2, 3);
	print_array_of_int(array2, 1);
	return 0;
}
```

Σχήμα 13

Αν τα a και b είναι arrays τότε δεν μπορούμε να χρησιμοποιήσουμε το συμβολισμό

```
a = b;
```

για να δώσουμε στα στοιχεία του a τις τιμές των αντίστοιχων στοιχείων του b· για να το επιτύχουμε αυτό θα πρέπει να δώσουμε σε ξεχωριστά σε κάθε στοιχείο του a την τιμή του στοιχείου του b που επιθυμούμε:

```
for (i = 0; i < a_size; i++)
	a[i] = b[i];
```

Τα strings στη C

'Ενα string στη C είναι στην πραγματικότητα μία σειρά από χαρακτήρες των οποίων το τέλος υποδηλώνεται από την ύπαρξη του χαρακτήρα '\0'. Μία σταθερά τύπου string ορίζεται όπως έχουμε ήδη δει (συμβολισμός "..."), ενώ για να δηλώσουμε μία μεταβλητή τύπου string θα τη δηλώσουμε ως array χαρακτήρων. Το μέγεθος του array θα είναι το μέγιστο πλήθος χαρακτήρων που θέλουμε να χωράνε στο array συν ένα για το χαρακτήρα '\0'. 'Ετσι η δήλωση

char my_string[50];

δηλώνει τη μεταβλητή *my_string* που χωράει 49 κανονικούς χαρακτήρες και το '\0'.

Μία δήλωση μεταβλητής τύπου string μπορεί να έχει και αρχικοποίηση (με τους περιορισμούς που προαναφέρθησαν). Αυτή μπορεί να γίνει:

α) όπως και στα υπόλοιπα arrays, και έτσι η δήλωση
char str[20] = {'h', 'e', 'l', 'l', 'o', '\0'};
δηλώνει το string str το οποίο χωράει μέχρι 19 χαρακτήρες και το χαρακτήρα '\0'. Τα έξι πρώτα στοιχεία του str θα αρχικοποιηθούν με τις κατάλληλες τιμές

β) με τον -πιο εύχρηστο- συμβολισμό
char str[20] = "Hello";
που είναι ισοδύναμη με αυτή στο (α). Παρατηρούμε ότι δεν είναι απαραίτητο να βάλουμε το χαρακτήρα '\0', γιατί αυτό το κάνει ο compiler.

'Οπως και στα άλλα arrays έτσι και στα strings μπορούμε σε μία αρχικοποιημένη δήλωση να παραλείψουμε το πλήθος στοιχείων του array και ο compiler θα το συμπεράνει από την έκφραση που χρησιμοποιείται για να γίνει η αρχικοποίηση.

Δεδομένου ότι τα strings είναι arrays δεν μπορούμε να χρησιμοποιήσουμε σ' αυτά τελεστές για να ελέγξουμε την ισότητα, να εκχωρήσουμε την τιμή ενός string σε κάποιο άλλο, κ.ο.κ. Για να κάνουμε αυτές τις λειτουργίες θα πρέπει να χρησιμοποιήσουμε συναρτήσεις που παρέχονται στη βασική βιβλιοθήκη της γλώσσας C. Οι πιο συχνά χρησιμοποιούμενες από αυτές είναι:

α) *strcpy(char str1[], char str2[])*, η οποία αντιγράφει το string *str2* στο string *str1*. Μετά την κλήση της συνάρτησης δηλαδή το str1 θα έχει τους ίδιους χαρακτήρες με το str2

β) *strncpy(char str1[], char str2[], int n)*, με την οποία αντιγράφουμε το πολύ *n* χαρακτήρες από το *str2* στο *str1*. Η συνάρτηση αυτή χρησιμοποιείται όταν δεν είμαστε βέβαιοι ότι το str1 έχει αρκετό χώρο διαθέσιμο για να αποθηκεύσει όλους τους χαρακτήρες που περιέχονται στο str2. 'Ετσι τη συνάρτηση strncpy θα τη χρησιμοποιούσαμε σε κάποια περίπτωση όπως η πιο κάτω:
char str1[20], str2[50];
...
strncpy(str1, str2, 19);
Η strncpy ενδέχεται να μην προσθέσει το χαρακτήρα '\0' στο τέλος του str1 αν το str2 έχει περισσότερους χαρακτήρες από n (εξαρτάται από την υλοποίηση του compiler). Για να είναι ασφαλής έτσι μία κλήση στη strncpy θα πρέπει να ακολουθείται από την εντολή
str1[n] = '\0'

γ) *strcat(char str1[], char str2[])* με την οποία προσθέτουμε στο τέλος του string str1 το string str2. 'Ετσι, αν το str1 έχει την τιμή *"Hello "* και το str2 την τιμή *"there!"* τότε μετά την κλήση της strcat το str1 θα έχει την τιμή *"Hello there!"*. Το str1 θα πρέπει να έχει αρκετό χώρο για να αποθηκεύσει τους έξτρα χαρακτήρες

δ) *strncat(char str1[], char str2[], int n)* με την οποία προσθέτουμε το πολύ *n* χαρακτήρες του str2 στο τέλος του str1, το οποίο πρέπει να έχει αρκετό χώρο για να αποθηκεύσει τους έξτρα χαρακτήρες. Στο νέο τέλος του str1 προστίθεται ο χαρακτήρας '\0'

ε) *size_t strlen(char s[])* η οποία επιστρέφει το πλήθος των χαρακτήρων που περιέχει το string s (ο χαρακτήρας '\0' δεν υπολογίζεται). Ο τύπος *size_t* ορίζεται από τον C compiler με βάση ήδη υπάρχοντες τύπους· συνήθως ορίζεται ως *int* ή *unsigned int* και είναι απόλυτα ασφαλές να εκχωρήσουμε το αποτέλεσμα της strlen (ή οποιασδήποτε άλλης συνάρτησης ή τελεστή επιστρέφει τύπο size_t) σε μία μεταβλητή τύπου int ή unsigned int

στ) *int strcmp(char str1[], char str2[])* που επιστρέφει έναν ακέραιο μικρότερο από το μηδέν αν το str1 είναι μικρότερο από το str2, 0 αν τα strings είναι ίσα και έναν ακέραιο μεγαλύτερο από το μηδέν αν το str1 είναι μεγαλύτερο από το str2. Η

σύγκριση στα strings γίνεται με λεξικογραφική σειρά, συγκρίνονται δηλαδή χαρακτήρα προς χαρακτήρα μέχρι να βρεθεί κάποιος διαφορετικός ή να φτάσουμε στο τέλος του ενός string. Αν έχει βρεθεί διαφορετικός χαρακτήρας τότε το string στο οποίο ανήκει ο μικρότερος από τους δύο χαρακτήρες είναι το μικρότερο. Αν φτάσουμε στο τέλος του ενός string, τότε αν ταυτόχρονα φτάσαμε στο τέλος του άλλου string, τα strings είναι ίσα· αν όχι, τότε το string στο τέλος του οποίου φτάσαμε είναι το μικρότερο

ζ) *int strncmp(char str1[], char str2[], int n)* η οποία είναι αντίστοιχη της strcmp με τη διαφορά ότι συγκρίνονται το πολύ *n* χαρακτήρες από τα strings

η) *int atoi(char s[])* η οποία μας επιστρέφει την ακέραια τιμή που αναπαριστά το string s. Αν, π.χ. το s έχει τιμή "1201" η atoi(s) θα επιστρέψει τον ακέραιο 1201. Δεν μπορούμε να επιτύχουμε το ίδιο αποτέλεσμα με το συμβολισμό *(int)s*. Το s μπορεί να περιέχει πρόσημο και κενά πριν και μετά από αυτό

θ) *long atol(char s[])* η οποία μας επιστρέφει την τύπου long τιμή που αναπαριστά το string s. Ισχύουν οι ίδιες παρατηρήσεις με την atoi

ι) *double atof(char s[])* που μας επιστρέφει την τύπου doyble τιμή που περιέχεται στο string s. Το s μπορεί να περιέχει οποιαδήποτε αναπαράσταση μιας double σταθεράς.

Για να χρησιμοποιηθούν οι συναρτήσεις *atoi*, *atol* και *atof* θα πρέπει να έχουμε πριν από την πρώτη χρησιμοποίησή τους την οδηγία *#include <stdlib.h>*, ενώ για τις υπόλοιπες την οδηγία *#include <string.h>*.

Πολυδιάστατα arrays

Εκτός arrays μιας διάστασης μπορούμε να δηλώσουμε και arrays περισσοτέρων από μιας διαστάσεων. Για να δηλώσουμε ένα array με δύο διαστάσεις χρησιμοποιούμε το συμβολισμό

```
τύπος_δεδομένου όνομα[πλήθος1][πλήθος2];
```

ο οποίος ορίζει ότι το array *όνομα* αποτελείται από $πλήθος_1$ γραμμές και $πλήθος_2$ στήλες. Στην τομή της γραμμής-ι και της στήλης-κ (για 0 σ ι σ $πλήθος_1$ - 1 και 0 σ κ σ $πλήθος_2$ - 1) βρίσκεται ένα δεδομένο τύπου *τύπος_δεδομένου* στο οποίο μπορούμε να αναφερθούμε με το συμβολισμό *όνομα[ι][κ]*. Στο σημείο αυτό χρειάζεται ιδιαίτερη προσοχή γιατί ο συμβολισμός *όνομα[ι, κ]* που χρησιμοποιείται σε άλλες γλώσσες προγραμματισμού δίνει αποτέλεσμα τελείως διαφορετικό από αυτό που θέλουμε χωρίς να είναι συντακτικά λάθος. Σε έναν τέτοιο συμβολισμό η έκφραση "*ι, κ*" που βρίσκεται μέσα στις τετράγωνες αγκύλες είναι ουσιαστικά ισοδύναμη με το "*κ*" (λόγω της λειτουργίας του τελεστή κόμμα) και έτσι ο συμβολισμός *όνομα[ι, κ]* είναι ουσιαστικά ισοδύναμος με τον *όνομα[κ]*. 'Οταν έχουμε ένα διδιάστατο array (έστω *πίνακας*) ο συμβολισμός *πίνακας[ι]* μπορεί να χρησιμοποιηθεί για να αναφερθούμε σε ολόκληρη την ι-γραμμή του array *πίνακας*, προκειμένου, π.χ. να την περάσουμε ως παράμετρο σε συνάρτηση. 'Ετσι αν ισχύει:

```
int table[4][5];
```

τότε μπορούμε να χρησιμοποιήσουμε τη συνάρτηση *print_array_of_int* που έχουμε ορίσει για να τυπώσουμε τα στοιχεία *table[3][0]* έως *table[3][4]*:

```
print_array_of_int(table[3], 5);
```

Μία δήλωση array δύο διαστάσεων μπορεί να έχει και αρχικοποίηση. Μία αρχικοποιημένη δήλωση ενός διδιάστατου array έχει τη μορφή

```
τύπος όνομα[πλήθος1][πλήθος2] = {τ1, τ2, ..., τν};
```

όπου το ν θα πρέπει να είναι μικρότερο ή ίσο του $πλήθος_1$ * $πλήθος_2$, και ο τύπος του κάθε ενός από τα $τ_ι$ θα πρέπει να συμφωνεί με τον τύπο *τύπος*. Το $τ_1$ θα είναι η αρχική τιμή του *όνομα[0][0]*, το $τ_2$ θα είναι η αρχική τιμή του *όνομα[0][1]* κ.ο.κ. Η

αρχική τιμή του *όνομα[1][0]* δίνεται από το $τ_{πλήθος1 + 1}$, ου *όνομα[1][1]* από το $τ_{πλήθος1 + 2}$ και του *όνομα[2][0]* από το $τ_{2 * πλήθος1 + 1}$. Αν δεν υπάρχουν αρκετά $τ_ι$ τότε τα στοιχεία του array που περισσεύουν δεν αρχικοποιούνται. Αν θέλουμε να αρχικοποιήσουμε μόνο τα πρώτα ν-στοιχεία σε μία σειρά ενός array και τα επόμενα $τ_ι$ να ισχύουν για την επόμενη σειρά του array, τότε βάζουμε τα ν-στοιχεία ανάμεσα σε *{}*. 'Ετσι η δήλωση

```
int a[3][3] = {{1, 2}, 3, 4, 5, 6};
```

ορίζει το διδιάστατο πίνακα a, με διαστάσεις 3 * 3, του οποίου κάθε στοιχείο είναι ακέραιος. Το a[0][0] θα έχει αρχική τιμή 1, το a[0][1] θα έχει αρχική τιμή 2 και το a[0][3] δεν αρχικοποιείται. Τα a[1][0], a[1][1], a[1][2] και a[2][0] θα έχουν αρχικές τιμές 3, 4, 5 και 6, αντίστοιχα, ενώ τα a[2][1] και a[2][2] δεν αρχικοποιούνται.

Στις αρχικοποιημένες δηλώσεις διδιάστατων arrays μπορούμε να παραλείψουμε την *πρώτη* διάσταση ο compiler θα καταλάβει το μήκος της από το πλήθος και τη δομή των τιμών που χρησιμοποιούνται για αρχικοποίηση. Την πρώτη διάσταση του array μπορούμε να παραλείψουμε και όταν περνάμε ένα διδιάστατο array ως παράμετρο σε συνάρτηση. Για διδιάστατα arrays χαρακτήρων μπορούμε να χρησιμοποιήσουμε strings για αρχικοποίηση. Στο σχήμα 14 υπάρχει η συνάρτηση print_month() που τυπώνει το πλήρες όνομα ενός μήνα όταν έχει δοθεί ο αύξων αριθμός του (1-12), ενώ στο σχήμα 15 η συνάρτηση find_max() βρίσκει το μέγιστο ακέραιο σε ένα διδιάστατο array 4 * 4.

```
char month_names[12][10] =
{
      "January", "February", "March", "April",
      "May", "June", "July", "August",
      "September", "October", "November", "December"
};

void print_month(int month)
{
      if ((month < 1) || (month > 12))
            printf("Illegal month.\n");
      else
            printf("%s\n", month_names[month - 1]);
}
```

Σχήμα 14

```
int find_max(int a[4][4])
{
      int i, j, max;

      for (i = 0, max = a[0][0]; i < 4; i++)
            for (j = 0; j < 4; j++)
                  if (a[i][j] > max)
                        max = a[i][j];
      return max;
}
```

Σχήμα 15

Κατά τρόπο ανάλογο με τα διδιάστατα arrays μπορούμε να ορίσουμε και πολυδιάστατα arrays. Η γενική σύνταξη ορισμού ενός πολυδιάστατου array είναι

```
τύπος όνομα[πλήθος1][πλήθος2]...[πλήθοςν];
```

Η δήλωση μπορεί να έχει αρχικοποιήσεις και στην περίπτωση αυτή μπορεί να παραλείπεται το πλήθος[1], το οποίο μπορεί επίσης να παραλείπεται και όταν περνάμε ένα τέτοιο array ως παράμετρο σε συνάρτηση. Τέλος και για πολυδιάστατα arrays τύπου χαρακτήρα (char) μπορεί να χρησιμοποιείται ο συμβολισμός ″...″.

8. Πέρασμα παραμέτρων σε συναρτήσεις - Pointers

Στο σχήμα 16 βλέπουμε ένα πρόγραμμα που αποτελείται από δύο συναρτήσεις, την inc_and_print() και τη main(). Η συνάρτηση inc_and_print() δέχεται μία ακέραια παράμετρο, την εκτυπώνει, κατόπιν την αυξάνει κατά 1 και την ξανατυπώνει, ενώ στη συνάρτηση main() ορίζεται μία ακέραια μεταβλητή *i* με αρχική τιμή 2. Η μεταβλητή αυτή εκτυπώνεται, κατόπιν περνά ως παράμετρος στη συνάρτηση inc_and_print() και τέλος εκτυπώνεται ξανά.

```
#include <stdio.h>

void inc_and_print(int j)
{
      printf("inc_and_print before increment: j = %d\n", j);
      j++;
      printf("inc_and_print after increment: j = %d\n", j);
}

int main()
{
      int i = 2;

      printf("main before call to inc_and_print: i = %d\n",i);
      inc_and_print(i);
      printf("main after call to inc_and_print: i = %d\n", i);
}
```

Σχήμα 16

Θα περιμέναμε ίσως το πρόγραμμα να εκτυπώσει τα ακόλουθα μηνύματα όταν θα το τρέχαμε:

```
main before call to inc_and_print: i = 2
inc_and_print before increment: j = 2
inc_and_print after increment: j = 3
main after call to inc_and_print: i = 3
```

Πίνακας 3

Στην πραγματικότητα όμως το τελευταίο μήνυμα είναι διαφορετικό:
main after call to inc_and_print: i = 2
Φαίνεται δηλαδή ότι η τιμή του i στη συνάρτηση main() **δεν επηρεάστηκε από την κλήση στη συνάρτηση inc_and_print()**. Αυτό συμβαίνει γιατί όταν η C περνά παραμέτρους σε κάποια συνάρτηση δημιουργεί **αντίγραφα** των παραμέτρων τα οποία θέτει στη διάθεση της καλούμενης συνάρτησης. Κατά συνέπεια όσες αλλαγές και αν κάνει η καλούμενη συνάρτηση στις μεταβλητές αυτές δεν θα έχουν καμία απολύτως επίδραση στις μεταβλητές της καλούσας συνάρτησης.
Εξαίρεση στον κανόνα αυτό είναι τα arrays για τα οποία **ποτέ** η C δεν δημιουργεί αντίγραφα. Αυτό γίνεται για λόγους απόδοσης (η αντιγραφή ενός array μπορεί να απαιτεί πολύ χρόνο και χώρο) και γιατί μερικές φορές το μέγεθος του array δεν είναι γνωστό (π.χ. όταν σε κάποια συνάρτηση δεν δηλώνουμε το μέγεθος του array που περνάμε ως παράμετρο). 'Ετσι οι αλλαγές που κάνει μία συνάρτηση στα στοιχεία ενός array που έλαβε ως παράμετρο **είναι** ορατές και στη συνάρτηση που προμήθευσε την παράμετρο αυτή (π.χ. συναρτήσεις strcpy() και strcat()). Φυσικά ως παράμετρος

περνά ένα στοιχείο ενός array (π.χ. ακέραιος, χαρακτήρας, πραγματικός), **θα δημιουργηθεί κανονικά αντίγραφο του στοιχείου**. Στο πρόγραμμα του σχήματος 17 φαίνονται οι δύο περιπτώσεις.

```
#include <stdio.h>

void inc_and_print(int j)
{
      printf("inc_and_print before increment: j = %d\n", j);
      j++;
      printf("inc_and_print after increment: j = %d\n", j);
}

void inc_and_print_array(int array[], int num_ints)
{
      int i;

      for (i = 0; i < num_ints; i++)
      {
            printf("%d ", array[i]);
            array[i]++;
            printf("%d ", array[i]);
      }
      printf("\n");
}

void inc_and_print_2d_array(int array[][2], int num_rows)
{
      int i;

      for (i = 0; i < num_rows; i++)
            inc_and_print_array(array[i]);
}

int a[3][2] = {{1, 2}, {10, 20}, {100, 200}};

int main()
{
      inc_and_print_2d_array(a, 3);
      inc_and_print_array(a[1]);
      inc_and_print(a[1][1]);
      printf("a[1][1] after inc_and_print: %d\n", a[1][1]);
      return 0;
}
```

Σχήμα 17

Αρκετές φορές όμως επιθυμούμε να μπορεί η συνάρτηση να τροποποιεί τα ορίσματά της κατά τρόπο που να είναι ορατός στη συνάρτηση που προμήθευσε την παράμετρο αυτή, έστω και αν η παράμετρος αυτή δεν είναι array (π.χ. η συνάρτηση scanf() που διαβάζει δεδομένα και τα αποθηκεύει σε μεταβλητές). Στις περιπτώσεις αυτές θα πρέπει η καλούσα συνάρτηση να προμηθεύσει στην καλούμενη *τη διεύθυνση της μεταβλητής* και όχι την ίδια τη μεταβλητή. Η διεύθυνση μιας μεταβλητής αποθηκεύεται στην καλούμενη συνάρτηση σε έναν *pointer* μέσω του οποίου μπορεί η συνάρτηση αυτή να προσπελάσει ή και να τροποποιήσει την τιμή της μεταβλητής. Αν ο pointer στην καλούμενη συνάρτηση έχει το όνομα p τότε μπορούμε να αναφερθούμε με το συμβολισμό **p* στην μεταβλητή στην οποία ο pointer δείχνει. Μπορούμε έτσι να πάρουμε την τιμή του:

```
printf("%d\n", *p);
```

ή να την αλλάξουμε:
```
*p = *p + 3;
```
Θα πρέπει να σημειώσουμε εδώ ότι ένας pointer δείχνει πάντα σε δεδομένα **συγκεκριμένου τύπου** και δεν λέμε απλά ότι είναι τύπου pointer αλλά **pointer σε δεδομένα τύπου x**. 'Ετσι η έκφραση **p* έχει επίσης τύπο, ο οποίος είναι x (ο pointer p του πιο πάνω παραδείγματος θεωρείται ότι είναι pointer σε δεδομένο τύπου int). Στη C το γεγονός ότι το p είναι pointer σε δεδομένο τύπου x συμβολίζεται με
```
x *a;
```
Αντίστροφα, από μία μεταβλητή a τύπου x μπορούμε να πάρουμε τη διεύθυνσή της (που είναι pointer σε δεδομένο τύπου x) χρησιμοποιώντας το συμβολισμό *&a*. Ο τελεστής *&* στη μορφή του αυτή θα πρέπει να ακολουθείται από μεταβλητή (που μπορεί να είναι και στοιχείο κάποιου array). Στο πρόγραμμα του σχήματος 18 φαίνεται η συνάρτηση inc_float που αυξάνει κατά ένα μία μεταβλητή τύπου float της οποίας τη διεύθυνση δέχεται ως παράμετρο και η συνάρτηση main() που περνάει τη διεύθυνση της μεταβλητής f ως παράμετρο στη συνάρτηση inc_float() και επίσης τυπώνει την τιμή του f πριν και μετά την κλήση.

```
#include <stdio.h>

void inc_float(float *f)
{
	*f += 1.0;
}

int main()
{
	float f = 32.897;

	printf("f before call to inc_float: %f\n", f);
	inc_float(&f);
	printf("f after call to inc_float: %f\n", f);
}
```

Σχήμα 18

Στο σημείο αυτό θα παρατηρήσουμε ότι αν και τα δεδομένα τύπου float μετατρέπονται σε double όταν περνάνε σε συναρτήσεις, δε συμβαίνει το ίδιο με τους pointers σε δεδομένα τύπου float για τους οποίους δεν γίνεται καμία απολύτως μετατροπή.

Μεταβλητές τύπου pointer

Αντικείμενα τύπου pointer μπορούμε να ορίσουμε σε οποιοδήποτε σημείο του προγράμματος επιτρέπεται η δήλωση μεταβλητών. Η δήλωση μιας μεταβλητής a που είναι pointer σε δεδομένα τύπου x έχει τη μορφή
```
x *a;
```
Προσοχή στο γεγονός ότι η δήλωση
```
x *a, b;
```
ορίζει τη μεταβλητή a που είναι pointer σε δεδομένα τύπου x και τη μεταβλητή b που είναι **τύπου x** και όχι pointer σε δεδομένα τύπου x. Αν θέλαμε και το b να είναι pointer η δήλωση θα έπρεπε να είναι
```
x *a, *b;
```
Μία μεταβλητή τύπου pointer (σε δεδομένα οποιουδήποτε τύπου) μπορεί να αρχικοποιείται, συνήθως με τη διεύθυνση κάποιας μεταβλητής ή με κάποια άλλη

μεταβλητή τύπου pointer ή με τη σταθερά NULL, η οποία θα αναλυθεί στη συνέχεια. 'Ετσι μπορούμε να έχουμε τις δηλώσεις:

```
int a;
int *ip = &a, *b = NULL;
int *ip1 = ip;
```

Από τη στιγμή που θα ορίσουμε μία μεταβλητή τύπου pointer σε δεδομένα τύπου x μπορούμε να εκχωρήσουμε σ' αυτή τη διεύθυνση κάποιας μεταβλητής τύπου x ή την τιμή κάποιας άλλης μεταβλητής ίδιου τύπου με τη x. Αν η μεταβλητή τύπου pointer που ορίσαμε δεν αρχικοποιείται, τότε η εκχώρηση αυτή θα πρέπει οπωσδήποτε να γίνει **πριν** χρησιμοποιήσουμε τον pointer για πρώτη φορά. Στην αντίθετη περίπτωση θα έχουμε άσχημα αποτελέσματα. Μία μεταβλητή τύπου pointer σε χαρακτήρα μπορεί να αρχικοποιηθεί και με το συμβολισμό

```
char *cp = "Initialized pointer";
```

με αποτέλεσμα ο pointer *cp* να δείχνει σε μία περιοχή στη μνήμη που περιέχει το string. 'Οταν έχουμε μία τέτοια δήλωση θα πρέπει να προσέχουμε γιατί αν αλλάξουμε την τιμή του pointer *cp* δεν θα μπορούμε πλέον να αναφερθούμε στο string.

Σε έναν pointer σε δεδομένα τύπου x μπορούμε επίσης να εκχωρήσουμε και ένα array με δεδομένα τύπου x. Το αποτέλεσμα μιας τέτοιας εκχώρησης είναι να πάρει ο pointer τη διεύθυνση του αρχικού (πρώτου) στοιχείου του array, οι συμβολισμοί δηλαδή

```
p = a;
```

και

```
p = &a[0]
```

όπου p είναι pointer σε δεδομένα τύπου x και a είναι array με δεδομένα τύπου x, είναι απολύτως ισοδύναμοι. Σε έναν pointer *p* μπορούμε να προσθέσουμε έναν ακέραιο *ι*. Κάτι τέτοιο θα πρέπει να το κάνουμε **μόνο** όταν είμαστε βέβαιοι ότι ο pointer p δείχνει σε κάποιο στοιχείο ενός array και το αποτέλεσμα της πρόσθεσης αυτής είναι ένας pointer ίδιου τύπου με τον p, που δείχνει στο ι-οστό στοιχείο του array που δείχνεται από τον p. 'Ετσι αν έχουμε:

```
p = a;
p1 = p + 6;
```

ο p1 θα δείχνει στο έβδομο στοιχείο του array a. Από έναν pointer μπορούμε ακόμη να αφαιρέσουμε έναν ακέραιο ι, και τότε το αποτέλεσμα της αφαίρεσης θα δείχνει ι-στοιχεία πριν από το σημείο που έδειχνε ο αρχικός pointer. 'Ετσι με τις εκχωρήσεις:

```
p = a + 5;
p1 = p - 3;
```

ο p και ο p1 θα δείχνουν στο έκτο και στο τρίτο στοιχείο του a, αντίστοιχα. Αν είμαστε βέβαιοι ότι δύο pointers δείχνουν στο ίδιο array (όπως οι p και p1 πιο πάνω) μπορούμε να αφαιρέσουμε τον έναν από τον άλλο. Το αποτέλεσμα της αφαίρεσης θα είναι ένας ακέραιος που θα δείχνει πόσα στοιχεία παρεμβάλλονται ανάμεσα σ' αυτά που δείχνονται από τους δύο pointers. Με τις εκχωρήσεις του πιο πάνω παραδείγματος θα έχουμε ότι:

```
p - p1 == 3
```

και, φυσικά,

```
p1 - p == -3
```

Στους pointers μπορούμε να χρησιμοποιήσουμε και τους τελεστές +=, -= (με μία ακέραια έκφραση στο δεξιό μέρος του τελεστή), ++ και --. Αν έχουμε το πρόγραμμα:

```
p = a;
p += 4;        (1)
p++;           (2)
p -=3;         (3)
p--;           (4)
```

τότε μετά την εντολή (1) ο p θα δείχνει στο πέμπτο στοιχείο του a, μετά την εντολή (2) στο έκτο, μετά την εντολή (3) στο τρίτο και μετά την εκτέλεση της εντολής (4) στο δεύτερο στοιχείο του a.
Η αντιστοιχία ανάμεσα σε pointers και arrays πηγαίνει ακόμη μακρύτερα. Στη C όλοι οι συμβολισμοί που μπορούν να χρησιμοποιηθούν για ένα array μπορούν να χρησιμοποιηθούν και για έναν pointer και αντίστροφα. 'Ετσι αν το a είναι array από δεδομένα τύπου x και το p είναι pointer σε δεδομένα τύπου x μπορούν να χρησιμοποιηθούν οι συμβολισμοί του πίνακα 4 (συμβολισμοί που εμφανίζονται στην ίδια γραμμή είναι ισοδύναμοι):

a[0]	*a	*(a + 0)
a[5]	*(a + 5)	
p = a	p = &a[0]	p = a + 0
p = &a[5]	p = a + 5	
p[2]	*(p + 2)	
p += 9	p = &p[9]	p = &*(p + 9)

Πίνακας 4

Παρατηρούμε ότι ο μοναδικός συμβολισμός για pointers που δεν μπορεί να χρησιμοποιηθεί για arrays είναι η εκχώρηση τιμής, δεν μπορούμε δηλαδή να εκχωρήσουμε κάποια τιμή στο όνομα ενός array (μπορούμε βέβαια να εκχωρήσουμε τιμή στα στοιχεία του). Αυτό συμβαίνει γιατί το array είναι ένα συμβολικό όνομα για ένα πλήθος μεταβλητών που βρίσκεται σταθερά σε κάποια περιοχή της μνήμης, ενώ ο pointer είναι απλά μια διεύθυνση μνήμης που μπορεί να δείχνει όπου θέλουμε. Μία άλλη διαφορά ανάμεσα στους pointers και τα arrays είναι ότι ένα array από τη στιγμή της δήλωσής του δεσμεύει χώρο στη μνήμη για τα στοιχεία του, ενώ με τη δήλωση ενός pointer δεσμεύεται χώρος μόνο για τον ίδιο τον pointer, και έτσι αυτός θα δείχνει σε κάποια τυχαία θέση στη μνήμη. Η ιδιότητα των pointers δέχονται εκχωρήσεις τιμών χρησιμοποιείται, μεταξύ άλλων, για να μπορούν συναρτήσεις να επιστρέφουν arrays, γεγονός που επιτυγχάνεται με το να επιστρέφεται η διεύθυνση του πρώτου στοιχείου του array που μας ενδιαφέρει.
Στο πρόγραμμα του σχήματος 19 φαίνεται η πλήρης αντιστοιχία μεταξύ pointers και arrays. Η συνάρτηση first_occurcene() επιστρέφει έναν pointer στην πρώτη θέση του string *str* στην οποία εμφανίζεται ο χαρακτήρας *c*, ενώ η last_occurence() ψάχνει επίσης για το χαρακτήρα *c*, επιστρέφοντας όμως τη θέση στην οποία εμφανίζεται για τελευταία φορά (που μπορεί να συμπίπτει με την πρώτη αν ο χαρακτήρας εμφανίζεται ακριβώς μία φορά). Και οι δύο συναρτήσεις επιστρέφουν τη σταθερά NULL αν ο χαρακτήρας αυτός δεν υπάρχει στο str. Η συνάρτηση strstat τυπώνει κάποια μηνύματα για την εμφάνιση του χαρακτήρα c στο str και η συνάρτηση main() καλεί τη συνάρτηση strstat() για το string *string* ή τμήματά του.

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

char *first_occurence(char *str, char c)
{
	for (; *str != '\0'; str++)
		if (*str == c)
			return str;
	return NULL;
}
```

```
char *last_occurence(char str[], char c)
{
	char *posn, *new_posn;

	if ((posn = first_occurence(str, c)) == NULL)
		return NULL;
	while (1)
	{
		new_posn = first_occurence(posn + 1, c);
		if (new_posn == NULL)
			return posn;
		else
			posn = new_posn;
	}
}

void strstat(char *s, char c)
{
	int i;
	char *s1, *s2;

	s1 = first_occurence(s, c);
	s2 = last_occurence(s, c);
	if (s1 == NULL)
	{
		printf("Character %c not found in string %s\n",
			c, s);
		return;
	}
	printf("Character %c:\n", c);
	printf("\tFound at position %d in string %s\n",
			s1 - s, s);
	printf("\tString after first occurence of %c: %s\n",
			c, s1);
	printf("\tLast occurence in string %s at position %d\n",
			s, s2 - s);
	printf("\tString after last occurence of %c: %s\n",
			c, s2);
	printf("\t%d characters between the two occurences\n",
			s2 - s1);
}

int main()
{
	char string[100];

	strcpy(string, "This is a string");
	strstat(string, 'i');
	strstat(string + 10, 'i');
	strstat(string, 'c');
	return 0;
}
```

Σχήμα 19

Η σταθερά NULL είναι ορισμένη στο αρχείο *stdio.h* (ενδέχεται να είναι ορισμένη και σε άλλα *.h* αρχεία) και είναι μία ειδική τιμή που χρησιμοποιείται στους pointers. Η C μας εγγυάται ότι ένας pointer που δείχνει σε δεδομένα που έχουν νόημα δεν έχει ποτέ την τιμή NULL, και έτσι η τιμή αυτή χρησιμοποιείται κυρίως από συναρτήσεις που επιστρέφουν pointers για να δείξουν συνθήκες λάθους ή μη ύπαρξης (όπως στο πιο πάνω παράδειγμα).

Pointers σε pointers - Arrays από pointers

Φυσικά αν θέλουμε να περάσουμε έναν pointer σε μία συνάρτηση και να μπορεί αυτή να τροποποιήσει την τιμή του κατά τρόπο που να είναι ορατός στη συνάρτηση που προμήθευσε την παράμετρο, τότε θα πρέπει να περάσουμε ως παράμετρο τη διεύθυνση του pointer, της οποίας ο τύπος είναι pointer σε pointer. Μία συνάρτηση που δέχεται ως παράμετρο έναν pointer σε pointer σε χαρακτήρα είναι η strtod(), που ορίζεται στο αρχείο *stdlib.h*. Το πρότυπό της είναι

```
double strtod(char *s, char **endp);
```

Η συνάρτηση αυτή λειτουργεί όπως η atof(), επιστρέφει δηλαδή την τιμή τύπου double που αναπαρίσταται στο string s, επιστρέφει όμως και μέσω του pointer *endp* το σημείο όπου η μετατροπή του string σε double σταμάτησε (λόγω του ότι ο χαρακτήρας δεν είναι παραδεκτός ή γιατί το string μετατράπηκε όλο). Στο σχήμα 20 φαίνεται ένα πρόγραμμα που χρησιμοποιεί τη συνάρτηση strtol().

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
      char s[100], *end;
      double d;

      strcpy(s, "3.2568e21 Invalid characters");
      d = strtod(s, &end);
      printf("Converted \"%s\" to double.\n", s);
      printf("Result = %g.\n", d);
      printf("String \"%s\" couldn't be converted.\n", end);
      return 0;
}
```

Σχήμα 20

Το πρόγραμμα, όταν το τρέξουμε, θα μας τυπώσει τα πιο κάτω μηνύματα:

```
Converted "3.2568e21 Invalid characters" to double.
Result = 3.2568e21.
String " Invalid characters" couldn't be converted.
```

Μπορούμε ακόμη να ορίσουμε μεταβλητές τύπου pointers σε pointers σε δεδομένα τύπου x χρησιμοποιώντας το συμβολισμό

```
x **p;
```

Μία τέτοια δήλωση μπορεί να αρχικοποιείται:

```
x *p0;
x **p1 = &p0;
```

Επειδή ο pointer σε pointer σε δεδομένα τύπου x παραμένει στην ουσία pointer, μπορούν να χρησιμοποιηθούν με αυτόν όλοι οι συμβολισμοί του πίνακα 4, με τον περιορισμό ότι το array a θα είναι πλέον **array από pointers σε δεδομένα τύπου x**. 'Ενα array από pointers σε δεδομένα τύπου x ορίζεται με τη δήλωση

```
x *a[πλήθος];
```

(το πλήθος μπορεί να παραλείπεται όταν το a είναι παράμετρος συνάρτησης ή όταν υπάρχει αρχικοποίηση). Arrays από pointers συνήθως ορίζονται για χαρακτήρες, γιατί πλεονεκτούν σε χώρο αποθήκευσης σε σχέση με τα διδιάστατα arrays. 'Ετσι η δήλωση

```
char names[][21] = {"Dick Jones", "Tom Soyer",
                    "Michael Gerstenhaber"};
```

απαιτούνται 21 * 3 = 63 bytes, ενώ για την αποθήκευση του ίδιου array δηλωμένου ως

```
char *names[] = ...;
```

απαιτούνται 11 + 10 + 21 + 12 = 54 bytes (42 bytes για την αποθήκευση των τριών strings + 12 bytes για την αποθήκευση των τριών pointers - υποθέσαμε pointers των τεσσάρων bytes, που είναι πολύ συνηθισμένοι). Η διαφορά αυτή μπορεί να είναι πολύ

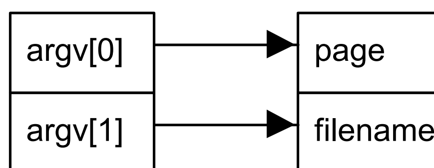
μεγαλύτερη όταν αυξάνει το πλήθος των strings και κυρίως όταν αυξάνει η διασπορά των μηκών τους. Στα διδιάστατα arrays θα πρέπει πάντα να κρατάμε σε κάθε στοιχείο χώρο αρκετό για να χωρέσει το μεγαλύτερο string, ενώ με τους pointers δεσμεύεται για κάθε string χώρος ίσος με το μήκος του, συν ένα για το χαρακτήρα '\0' συν τέσσερα για την αποθήκευση του pointer.
'Ενα array από pointers σε χαρακτήρες ορίζει και η ίδια η C και το περνάει ως δεύτερη παράμετρο στη συνάρτηση main(). Στην παράμετρο αυτή δίνουμε τυπικά το όνομα *argv*, ενώ η πρώτη παράμετρος είναι ένας ακέραιος στον οποίο τυπικά δίνουμε το όνομα *argc*. Οι δύο αυτές παράμετροι της συνάρτησης main(), της οποίας το πρότυπο είναι, τελικά

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

χρησιμοποιούνται για να μπορέσουμε να προσπελάσουμε ορίσματα που ο χρήστης ήθελε να περάσει στο πρόγραμμά μας. Αν, για παράδειγμα, το πρόγραμμά μας είχε το όνομα *page* και είχε ως σκοπό να τυπώνει κατά σελίδες τα περιεχόμενα ενός αρχείου στην οθόνη, θα ήταν λογικό ο χρήστης να το καλεί από το λειτουργικό σύστημα με μία εντολή της μορφής

```
page filename
```

Μία τέτοια κλήση θα είχε ως αποτέλεσμα η παράμετρος argc της συνάρτησης main() να πάρει την τιμή 2, και η παράμετρος argv να είναι ένα array από δύο pointers σε χαρακτήρες που φαίνεται στο σχήμα 21:



Σχήμα 21

Στο σχήμα 22 φαίνεται ένα πρόγραμμα, που αποτελείται μόνο από τη συνάρτηση main(), το οποίο τυπώνει τα ορίσματα που προμήθευσε ο χρήστης κατά την κλήση του προγράμματος από το λειτουργικό σύστημα.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
	int i;

	printf("User supplied %d arguments.\n", argc);
	for (i = 0; i < argc; i++)
		printf("Argument %d = %s\n", i, argv[i]);
	return 0;
}
```

Σχήμα 22

Pointers σε συναρτήσεις

Εκτός από pointers σε δεδομένα μπορούμε να ορίσουμε και pointers σε συναρτήσεις.
'Ενας pointer σε συνάρτηση ορίζεται με μία δήλωση της μορφής

```
τύπος (*όνομα)(τ1, τ2, ..., τν)
```

Η πιο πάνω δήλωση ορίζει τον pointer *όνομα* που δείχνει σε μία συνάρτηση η οποία επιστρέφει κάποιο δεδομένο τύπου *τύπος* και δέχεται ν παραμέτρους, η πρώτη από τις οποίες είναι τύπου $τ_1$, η δεύτερη $τ_2$, κ.ο.κ. Η λίστα των τύπων μέσα στις παρενθέσεις

μπορεί να παραλειφθεί, αλλά είναι καλό να δίνεται προκειμένου να έχουμε έλεγχο τύπων των παραμέτρων κατά την κλήση της συνάρτησης. Οι παρενθέσεις που περιβάλουν το *όνομα* είναι απαραίτητες γιατί ο συμβολισμός

```
τύπος *όνομα(τ1, τ2, ..., τν)
```

δεν ορίζει pointer σε συνάρτηση, αλλά είναι το πρότυπο μιας συνάρτησης που ονομάζεται *όνομα*, επιστρέφει δεδομένο τύπου *τύπος* * (δηλ. pointer σε δεδομένα τύπου *τύπος*) και δέχεται ν παραμέτρους, η πρώτη από τις οποίες είναι τύπου $τ_1$, κ.λ.π.

Για να ορίσουμε ότι κάποιος pointer σε συνάρτηση δείχνει σε κάποια συγκεκριμένη συνάρτηση χρησιμοποιούμε το συμβολισμό

```
function_pointer = function_name;
```

(όπου *function_pointer* είναι ένας pointer σε συνάρτηση και *function_name* το **όνομα** κάποιας συνάρτησης, χωρίς να ακολουθείται από παρενθέσεις. Φυσικά επιτρέπεται μία εκχώρηση της μορφής

```
function_pointer1 = function_pointer2;
```

Για να καλέσουμε τώρα μία συνάρτηση που δείχνεται από έναν pointer σε συνάρτηση χρησιμοποιούμε το συμβολισμό

```
(*function_pointer)(p1, p2, ..., pν);
```

'Οταν χρησιμοποιούμε το συμβολισμό αυτό θα πρέπει να είμαστε απόλυτα βέβαιοι ότι ο pointer *function_pointer* δείχνει σε κάποια συνάρτηση, αλλιώς τα αποτελέσματα θα είναι καταστροφικά. Στο πρόγραμμα του σχήματος 23 φαίνεται η συνάρτηση sum, που δέχεται δύο παραμέτρους *a* και *b* τύπου double και έναν pointer *f* σε συνάρτηση που δέχεται ως όρισμα έναν double και επιστρέφει έναν double. Η συνάρτηση επιστρέφει την τιμή του ορισμένου ολοκληρώματος από a έως b της συνάρτησης που δείχνεται από τον pointer f, υπολογισμένη κατά αριθμητικό τρόπο. Η συνάρτηση main() καλεί τη συνάρτηση sum() να υπολογίσει το ορισμένο ολοκλήρωμα της exp(x) (e^x) από 0 έως 2 και του λογαρίθμου με βάση 10 (log10(x)) από 10 έως 20 και εκτυπώνει τα αποτελέσματα.

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

double sum(double a, double b, double (*f)(double))
{
      double step = (b - a) / 100, value = 0.0;

      while (a < b)
      {
            value += (*f)(a + (step / 2)) * step;
            a += step;
      }
      return value;
}

int main(int argc, char *argv[])
{
      printf("sum(0, 2, exp) = %g\n", sum(0.0, 2.0, exp));
      printf("sum(10, 20, log10) = %g\n", sum(10, 20, log10));
      return 0;
}
```

Σχήμα 23

9. Είσοδος/'Εξοδος στη C

Μέχρι τώρα έχουμε δει τη συνάρτηση που χρησιμοποιείται περισσότερο από κάθε άλλη για έξοδο σε ένα πρόγραμμα C που είναι η printf(). Σε αντιστοιχία με την printf() υπάρχει η συνάρτηση *scanf()* που χρησιμοποιείται για διάβασμα δεδομένων, δηλαδή είσοδο. Η βασική σύνταξη της scanf() είναι όμοια με αυτή της printf(), έχουμε δηλαδή

```
scanf(format, p1, p2, ..., pν);
```

με τη διαφορά ότι κάθε μία από τις παραμέτρους p_ι θα πρέπει να είναι *pointer* σε δεδομένα (ή array, αφού, όπως έχουμε δει, η χρήση arrays και pointers είναι σχεδόν ταυτόσημη). Η scanf λειτουργεί κατά τον ακόλουθο τρόπο: Διαβάζει το format από τον πρώτο χαρακτήρα προς τον τελευταίο. Αν ο επόμενος χαρακτήρας είναι κενό (space) ή tab η scanf() τον αγνοεί, ενώ αν δεν είναι *'%'*, τότε περιμένει να βρει στην είσοδο αυτόν ακριβώς το χαρακτήρα. Αν ο επόμενος χαρακτήρας είναι %, τότε οι επόμενοι από αυτόν χαρακτήρες καθορίζουν τον τύπο του δεδομένου που θα διαβαστεί. Για να διαβάσει το δεδομένο η scanf αγνοεί κενά, tabs και newlines και κατόπιν διαβάζει τους χαρακτήρες μέχρι το επόμενο κενό, tab ή newline ή μέχρι να βρει κάποιο χαρακτήρα που δεν είναι παραδεκτός για το συγκεκριμένο τύπο δεδομένου (π.χ. γράμμα όταν διαβάζει ακέραιο). Κατόπιν μετατρέπει τους χαρακτήρες που διάβασε στον κατάλληλο τύπο, αν αυτό είναι απαραίτητο, και αποθηκεύει το αποτέλεσμα στη διεύθυνση που ορίζει ο αντίστοιχος pointer. Αν σε κάποιο σημείο η scanf() δεν βρει κάποιο χαρακτήρα που περίμενε ότι θα υπάρχει, τότε τερματίζει και το υπόλοιπο format αγνοείται. Οι χαρακτήρες που μπορούν να χρησιμοποιηθούν μετά το *%* είναι όμοιοι με την printf() με τις εξής προσθήκες/διαφορές:

α) οι χαρακτήρες *f*, *g* και *e* σημαίνουν δείκτη σε δεδομένο τύπου *float* και όχι double. Αν θέλουμε να διαβάσουμε κάποιο δεδομένο τύπου double θα πρέπει να τοποθετήσουμε ανάμεσα στο *%* και στους χαρακτήρες αυτούς (όποιον από τους τρεις εμφανίζεται) το χαρακτήρα *l*, ενώ αν θέλουμε να διαβάσουμε δεδομένο τύπου long double θα πρέπει να τοποθετήσουμε το χαρακτήρα *L*. Και οι τρεις χαρακτήρες μπορούν να διαβάσουν οποιαδήποτε αναπαράσταση του πραγματικού

β) ο χαρακτήρας *c* ορίζει ότι η scanf δεν θα αγνοεί τα αρχικά κενά, και έτσι χρησιμοποιείται για να διαβάσουμε τον επόμενο χαρακτήρα από την είσοδο, ανεξάρτητα με το αν είναι κενό ή όχι

γ) ο συμβολισμός [...] χρησιμοποιείται για να διαβάσουμε ένα string που αποτελείται από τους χαρακτήρες που περικλείονται μέσα στις τετράγωνες αγκύλες. Μπορούν να χρησιμοποιηθούν *περιοχές* (ranges), δηλαδή συμβολισμοί της μορφής A-Z, που περιλαμβάνει όλους τους κεφαλαίους χαρακτήρες του αγγλικού αλφαβήτου. 'Ετσι η κλήση

   ```
   scanf("%[A-Za-z ]", s);
   ```

   (όπου το s είναι pointer σε χαρακτήρα) θα διαβάσει το επόμενο string που αποτελείται από κεφαλαίους και μικρούς χαρακτήρες του λατινικού αλφαβήτου και κενά

δ) μπορεί να χρησιμοποιηθεί ο χαρακτήρας *n* που αποθηκεύει στον αντίστοιχο pointer, που πρέπει να είναι pointer σε ακέραιο, το πλήθος των χαρακτήρων που έχουν διαβαστεί μέχρι το σημείο εκείνο.

Ανάμεσα από το *%* και τον προσδιοριστή τύπου μπορούμε να τοποθετήσουμε έναν ακέραιο, που δείχνει το μέγιστο πλήθος χαρακτήρων που θα διαβάσει η scanf για το συγκεκριμένο πεδίο. Ο αριθμός αυτός χρησιμοποιείται συνήθως με τους τύπους *s*, *c*

και το συμβολισμό *[...]*. Με το χαρακτήρα *c* ο αριθμός αυτός ορίζει ότι θα διαβαστούν *ακριβώς* τόσοι χαρακτήρες, αντί για έναν που διαβάζεται κανονικά, εκτός αν υπάρξει κάποια συνθήκη λάθους.
Αν θέλουμε απλά να διαπιστώσουμε την ύπαρξη ενός δεδομένου κατάλληλου τύπου στην είσοδο και δεν ενδιαφερόμαστε για την τιμή του, τότε θα πρέπει να τοποθετήσουμε αμέσως μετά από το χαρακτήρα *%* το χαρακτήρα *. Για το δεδομένο αυτό δεν πρέπει να παρέχουμε κάποιον pointer.
Η scanf() επιστρέφει το πλήθος των δεδομένων που διαβάστηκαν και αποθηκεύθηκαν στους αντίστοιχους pointers.

Παραδείγματα στη scanf()
Για τις μεταβλητές που χρησιμοποιούνται στα πιο κάτω παραδείγματα ισχύουν οι εξής δηλώσεις:
```
char s[30];
int i;
float f;
double d;
```
Παράδειγμα 1: *scanf("%d", &i);* που διαβάζει από την είσοδο χαρακτήρες που αντιστοιχούν σε ακέραιο (μπορεί να έχει πρόσημο), υπολογίζει τον αντίστοιχο ακέραιο και τον αποθηκεύει στη μεταβλητή i. Αν στην είσοδο πληκτρολογήσουμε τους χαρακτήρες "-10322<ENTER>" ο i θα πάρει την τιμή -10322 και η scanf() θα επιστρέψει 1, ενώ το ίδιο θα συμβεί αν πληκτρολογήσουμε τους χαρακτήρες " <TAB><ENTER> -10322x0". Αν στην είσοδο πληκτρολογήσουμε όμως τους χαρακτήρες "abcd" τότε η scanf() θα επιστρέψει 0 και η τιμή του i θα παραμείνει αμετάβλητη.
Παράδειγμα 2: *scanf("%s sold an item costing %d", s, &i);* που θα ψάξει στην είσοδο για ένα string χωρίς κενά, το οποίο θα αποθηκεύσει στο s, κατόπιν για τους χαρακτήρες *sold an item costing* και τέλος για έναν ακέραιο, του οποίου η τιμή θα αποθηκευθεί στη μεταβλητή *i*. 'Ετσι, αν στην είσοδο πληκτρολογήσουμε τους χαρακτήρες "John sold an item costing 1200" στο string *s* θα αποθηκευθούν οι χαρακτήρες *John* (και ο χαρακτήρας '\0'), στον ακέραιο *i* η τιμή 1200 και η scanf() θα επιστρέψει την τιμή 2. Αν, όμως, πληκτρολογήσουμε στην είσοδο τους χαρακτήρες "John Black sold an item costing 1200" στο string s θα αποθηκευθούν οι χαρακτήρες *John* και η scanf() θα τερματίσει, γιατί περιμένει να βρει στην είσοδο το χαρακτήρα *B* ενώ περίμενε το χαρακτήρα *s*. Η scanf() θα επιστρέψει τιμή 1.
Παράδειγμα 3: *scanf("%d%[A-Za-z]%lf", &i, s, &d);* η οποία διαβάζει από την είσοδο έναν ακέραιο, που αποθηκεύεται στη μεταβλητή i, ένα string που αποτελείται από λατινικούς χαρακτήρες και κενά, το οποίο αποθηκεύεται στο string s και έναν πραγματικό (double) που αποθηκεύεται στη μεταβλητή *d*. Αν στην είσοδο πληκτρολογήσουμε τους χαρακτήρες "100 Hello there John 30e+1", στις μεταβλητές i, s και d θα αποθηκευθούν, αντίστοιχα, οι τιμές 100, "Hello there John " και 300.0, και η scanf() θα επιστρέψει 2. Αν στην είσοδο πληκτρολογούσαμε τους χαρακτήρες "100 Hello there John! 30e+1", στις μεταβλητές i και s θα αποθηκευόταν οι ίδιες τιμές με την προηγούμενη περίπτωση, αλλά λόγω του ότι το θαυμαστικό δεν μπορεί να περιληφθεί στο string αλλά δεν αποτελεί και παραδεκτό χαρακτήρα για έναν πραγματικό, η scanf() θα τερματιζόταν επιστρέφοντας 2 και αφήνοντας την τιμή του d άθικτη. Αν στην είσοδο είχαμε τους χαρακτήρες "100 ! 33.4" θα διαβαζόταν μόνο το 100, που θα αποθηκευόταν στη μεταβλητή i και η scanf θα επέστρεφε 1.
Παράδειγμα 4: *scanf("%6s%d", s, &i);* που διαβάζει ένα string με μήκος το πολύ έξι χαρακτήρες από την είσοδο και έναν ακέραιο. 'Ετσι, αν πληκτρολογήσουμε τους

χαρακτήρες "John 2000" στο s θα αποθηκευθεί η τιμή *John* και στον i η τιμή 2000, και η scanf θα επιστρέψει 2. Αν, αντίθετα, δώσουμε "Michael 120", τότε στο s θα αποθηκευθούν οι χαρακτήρες *Michae* και η scanf() θα τερματίσει επιστρέφοντας 1, γιατί ο χαρακτήρας *l* που ακολουθεί δεν είναι παραδεκτός για έναν ακέραιο. Αν στην είσοδο δώσουμε "Yes No 100", τότε στο s θα αποθηκευθεί η τιμή *Yes* και η scanf() θα τερματίσει επιστρέφοντας 1, γιατί ο χαρακτήρας *N* που ακολουθεί δεν είναι παραδεκτός για έναν ακέραιο.

Παράδειγμα 5: *scanf("%6c%d", s, &i);* που διαβάζει τους έξι επόμενους χαρακτήρες ανεξάρτητα από την τιμή τους και τους αποθηκεύει στο string s, **χωρίς να προσθέτει το χαρακτήρα '\0'** και έναν ακέραιο που τον αποθηκεύει στη μεταβλητή i. Αν στην είσοδο είχαμε "John 120", τότε στο s θα αποθηκευόταν οι χαρακτήρες "*John 1*" στον i η τιμή 20 και η scanf() θα επέστρεφε 2. Στην περίπτωση που είχαμε στην είσοδο "Yes No 100" στο s θα αποθηκευόταν οι χαρακτήρες "*Yes No*", στον i η τιμή 100 και η scanf() θα επέστρεφε 2. Αν, τέλος, δώσουμε "Michael 120", τότε στο s θα αποθηκευθούν οι χαρακτήρες "*Michae*" και η scanf() θα τερματίσει επιστρέφοντας 1, γιατί ο χαρακτήρας *l* που ακολουθεί δεν είναι παραδεκτός για έναν ακέραιο.

Παράδειγμα 6: *scanf("%d%*8c%f", &i, &f);* η οποία διαβάζει έναν ακέραιο που αποθηκεύεται στη μεταβλητή i, κατόπιν οκτώ χαρακτήρες οι οποίοι δεν αποθηκεύονται (προσέξτε ότι δεν υπάρχει και αντίστοιχος pointer) και έναν πραγματικό (τύπου float) που αποθηκεύεται στη μεταβλητή f. Αν στην είσοδο δώσουμε "100ABCDEFGH1.1e-2" στη μεταβλητή i θα αποθηκευθεί η τιμή 100 στη μεταβλητή f η τιμή 0.011 και η scanf() θα επιστρέψει 2.

'Αλλες συναρτήσεις εισόδου/εξόδου

Οι συναρτήσεις printf()/scanf() δεν είναι οι μόνες που παρέχει η C για είσοδο/έξοδο. Στο αρχείο *stdio.h* ορίζονται τα πρότυπα των πιο κάτω συναρτήσεων:

α) *int getchar(void)* που διαβάζει τον επόμενο χαρακτήρα από την είσοδο, τον οποίο και επιστρέφει. 'Οταν η είσοδος τελειώσει, δεν υπάρχουν δηλαδή άλλοι χαρακτήρες, η συνάρτηση επιστρέφει τη σταθερά EOF, που είναι ορισμένη στο αρχείο *stdio.h* και έχει τιμή διαφορετική από κάθε παραδεκτό χαρακτήρα

β) *char *gets(char *s)* που διαβάζει ένα string από την είσοδο μέχρι το πρώτο newline, το οποίο δεν αποθηκεύεται στο string, αλλά αντικαθίσταται με το χαρακτήρα '\0'. Η gets() επιστρέφει το s, αν το διάβασμα έγινε κανονικά ή τη σταθερά NULL αν δεν υπήρχαν χαρακτήρες να διαβαστούν

γ) *int putchar(int c)* που γράφει στην έξοδο το χαρακτήρα c· επιστρέφει το χαρακτήρα c ή τη σταθερά EOF σε περίπτωση λάθους

δ) *int puts(char *s)* που γράφει στην έξοδο το string s ακολουθούμενο από ένα newline. Επιστρέφει τη σταθερά EOF αν έγινε λάθος ή κάποιο μη αρνητικό ακέραιο αν η λειτουργία υπήρξε επιτυχής.

Το πρόγραμμα του σχήματος 24 μετράει τους χαρακτήρες, τις λέξεις και τις γραμμές που δόθηκαν στην είσοδο (ως λέξη ορίζεται μία ακολουθία από μη κενούς χαρακτήρες).

```
#include <stdio.h>

int main(int argc, char *argv)
{
	int chars = 0, words = 0, lines = 0, c;
	int state = 0;

	while ((c = getchar()) != EOF)
	{
```

```
        chars++;
        switch (c)
        {
              case '\n':
                    lines++;
                    state = 0;
                    break;
              case ' ':
              case '\t':
                    state = 0;
                    break;
              default:
                    if (state == 0)
                    {
                          state = 1;
                          words++;
                    }
        }
  }
  printf("\n%d lines, %d words, %d characters\n", lines,
              words, chars);
  return 0;
}
```

Σχήμα 24

Τα αρχεία στη C

Ο τρόπος με τον οποίο διαβάζουμε ή γράφουμε δεδομένα σε ένα αρχείο μέσα από τη C δεν διαφέρει από τον τρόπο με τον οποίο διαβάζουμε και γράφουμε δεδομένα από την είσοδο ή στην έξοδο αντίστοιχα. Πριν όμως μπορέσουμε να διαβάσουμε ή να γράψουμε δεδομένα σε κάποιο αρχείο πρέπει να δημιουργήσουμε μία αντιστοιχία ανάμεσα σε κάποιο δεδομένο του προγράμματος και στο αρχείο αυτό. Η αντιστοιχία δημιουργείται με τη συνάρτηση *fopen()* η οποία παίρνει δύο παραμέτρους τύπου *char* *. Η πρώτη από τις παραμέτρους αντιστοιχεί στο όνομα του αρχείου, η μορφή του οποίου καθορίζεται από το λειτουργικό σύστημα στο οποίο βρισκόμαστε, και η δεύτερη καθορίζει τον τρόπο με τον οποίο θα προσπελάσουμε το αρχείο και μπορεί να έχει τις ακόλουθες τιμές:

α) *"r"* που δηλώνει ότι θέλουμε να διαβάσουμε το αρχείο
β) *"w"* που δηλώνει ότι θέλουμε να δημιουργήσουμε ένα νέο αρχείο στο οποίο θα γράψουμε δεδομένα. Αν το αρχείο υπάρχει, καταστρέφεται, χάνονται δηλαδή τα περιεχόμενά του
γ) *"a"* που δηλώνει ότι θέλουμε να προσθέσουμε δεδομένα στο αρχείο. Αν το αρχείο υπάρχει τα περιεχόμενά του δεν καταστρέφονται και τα νέα δεδομένα προστίθενται στο τέλος του, ενώ αν δεν υπάρχει δημιουργείται
δ) *"r+"* που ανοίγει το αρχείο όπως και το "r" αλλά μας επιτρέπει και να γράψουμε στο αρχείο, με αποτέλεσμα να αλλάζουμε τα δεδομένα που υπήρχαν εκεί, αν η εγγραφή γίνεται στο μέσον του αρχείου ή να προστίθενται στο αρχείο, αν η εγγραφή γίνεται στο τέλος του αρχείου
ε) *"w+"* που ανοίγει το αρχείο όπως και το "w", αλλά μας επιτρέπει επίσης να διαβάσουμε το αρχείο
στ) *"a+"* που ανοίγει το αρχείο όπως και το "a", αλλά μας επιτρέπει επίσης να διαβάσουμε το αρχείο. Οι εγγραφές δεδομένων γίνονται πάντα στο τέλος του αρχείου

Η C ανοίγει όλα τα αρχεία με την πεποίθηση ότι περιέχουν κείμενο, και σε ένα αρχείο που έχει ανοιχθεί κατ' αυτό τον τρόπο γίνονται συνήθως μεταφράσεις (σε μερικά λειτουργικά συστήματα -π.χ. DOS- ο χαρακτήρας '\n' μετατρέπεται σε "\n\r" κατά την έξοδο, ενώ κατά την είσοδο γίνεται η αντίστροφη μετατροπή, σε άλλα -π.χ. Unix- ο χαρακτήρας '\0' μπορεί να σημαίνει το τέλος του αρχείου κ.λ.π.). Αν θέλουμε να ορίσουμε ότι δεν πρέπει να γίνονται αυτές οι μεταφράσεις στο αρχείο που ανοίγουμε πρέπει στο τέλος της δεύτερης παραμέτρου να προσθέσουμε το χαρακτήρα *b* (π.χ. *"r+b"*).

Η συνάρτηση *fopen()* επιστρέφει ένα αντικείμενο τύπου *FILE ** (ο τύπος αυτός είναι ορισμένος στο αρχείο stdio.h), το οποίο μπορούμε να το περάσουμε στη συνέχεια ως παράμετρο σε συναρτήσεις για να γράψουμε ή να διαβάσουμε δεδομένα από το αρχείο αυτό. Αν το άνοιγμα του αρχείου ήταν αδύνατο (π.χ. αν θέλαμε να ανοίξουμε για ανάγνωση ένα αρχείο που δεν υπήρχε) τότε η fopen() επιστρέφει τη σταθερά NULL. Για να ανοίξουμε έτσι το αρχείο "data" για να διαβάσουμε δεδομένα θα χρησιμοποιήσουμε τον κώδικα:

```
...
FILE *infile;
infile = fopen("data", "r");
if (infile == NULL)
{
      printf("Error opening file data for input\n");
      exit(2);
}
```

(η συνάρτηση exit είναι ορισμένη στο αρχείο *stdlib.h* και τερματίζει το πρόγραμμα επιστρέφοντας στο λειτουργικό σύστημα τον ακέραιο που δέχεται ως παράμετρο).

Για να διαβάσουμε ή να γράψουμε δεδομένα σε ένα αρχείο που έχουμε ανοίξει (και υπό την προϋπόθεση ότι ο τρόπος που το ανοίξαμε επιτρέπει την αντίστοιχη λειτουργία) μπορούμε να χρησιμοποιήσουμε τις πιο κάτω συναρτήσεις που είναι ορισμένες στο αρχείο stdio.h:

α) *int fprintf(FILE *fp, char *format, ...)* η οποία λειτουργεί όπως ακριβώς η printf() με τη μοναδική διαφορά ότι δέχεται μία επιπλέον παράμετρο τύπου *FILE ** που ορίζει σε ποιο αρχείο θα τοποθετήσει τους χαρακτήρες που η printf θα μας τύπωνε στην οθόνη. Η fprintf() επιστρέφει το πλήθος των χαρακτήρων που γράφηκαν στο αρχείο ή τη σταθερά EOF αν έγινε λάθος

β) *int fscanf(FILE *fp, char *format, ...)* η οποία λειτουργεί όπως ακριβώς η scanf() με τη μοναδική διαφορά ότι δέχεται μία επιπλέον παράμετρο τύπου *FILE ** που ορίζει από ποιο αρχείο διαβάσει τα δεδομένα που η scanf() θα διάβαζε από το πληκτρολόγιο. Η fscanf επιστρέφει το πλήθος των πεδίων που αποθηκεύθηκαν στους αντίστοιχους pointers ή τη σταθερά EOF αν φτάσαμε στο τέλος του αρχείου πριν γίνει οποιαδήποτε αποθήκευση σε pointer

γ) *int fgetc(FILE *fp)* που διαβάζει και επιστρέφει τον επόμενο χαρακτήρα από το αρχείο fp. Επιστρέφει EOF αν φτάσαμε στο τέλος του αρχείου

δ) *int fputc(int c, FILE *fp)* που γράφει το χαρακτήρα c στο αρχείο fp. Επιστρέφει το χαρακτήρα c ή EOF αν έγινε λάθος

ε) *char *fgets(char *s, int n, FILE *fp)* που διαβάζει χαρακτήρες από το αρχείο fp και τους αποθηκεύει στο string s, μέχρι να βρεθεί newline ('\n') ή να συμπληρωθούν n - 1 χαρακτήρες. Στο τέλος του s προστίθεται ο χαρακτήρας '\0', ενώ αν έχει βρεθεί ο χαρακτήρας '\n' συμπεριλαμβάνεται και αυτός στο string. Η fgets() επιστρέφει NULL, αν έγινε λάθος κατά την είσοδο, ή το s, αν η λειτουργία υπήρξε επιτυχής

στ) *int fputs(char *s, FILE *fp)* που λειτουργεί όπως ακριβώς η *puts()* αλλά το string s, καθώς και το newline γράφεται στο αρχείο fp, αντί στην έξοδο

ζ) *int putc(FILE *fp, int c)* και *int getc(FILE *fp)* που είναι απολύτως ισοδύναμες με τις fputc() και getc(), αντίστοιχα

η) *int ungetc(int c, FILE *fp)* η οποία τοποθετεί τον χαρακτήρα c 'πίσω' στο αρχείο fp για να διαβαστεί από την επόμενη εντολή εισόδου που θα αφορά το αρχείο αυτό. Ο χαρακτήρας c δεν μπορεί να είναι το EOF, και μπορούμε να καλέσουμε, στη γενική περίπτωση, μόνο μία φορά τη συνάρτηση αυτή χωρίς να καλέσουμε ενδιάμεσα κάποια συνάρτηση εισόδου (που θα διαβάσει κάποιο χαρακτήρα).

'Οταν έχουμε τελειώσει τις λειτουργίες που θέλαμε να κάνουμε με ένα αρχείο, τότε μπορούμε να διακόψουμε τη σύνδεση του FILE * που μας επέστρεψε η fopen() χρησιμοποιώντας τη συνάρτηση fclose(), που παίρνει ως παράμετρο τον FILE * του οποίου τη σύνδεση με το αρχείο θέλουμε να διακόψουμε και επιστρέφει 0 αν η λειτουργία ήταν επιτυχής και EOF στην αντίθετη περίπτωση. Η fclose() θα πρέπει να καλείται πάντα, ειδικά για αρχεία στα οποία έχουν γραφεί δεδομένα, γιατί υπάρχει κίνδυνος τα δεδομένα αυτά να μην αποθηκευθούν μόνιμα στο αρχείο.

Μπορούμε να αλλάξουμε το όνομα ενός αρχείου στο δίσκο χρησιμοποιώντας τη συνάρτηση rename() που δέχεται δύο παραμέτρους, η πρώτη από τις οποίες είναι το τρέχον όνομα του αρχείου και η δεύτερη είναι το νέο όνομα που θέλουμε να του δώσουμε. Η rename() επιστρέφει 0 σε περίπτωση επιτυχίας. 'Εχουμε ακόμη τη δυνατότητα να διαγράψουμε ένα αρχείο χρησιμοποιώντας τη συνάρτηση remove() που δέχεται ως παράμετρο το όνομα του αρχείου που θέλουμε να διαγράψουμε. Η συνάρτηση remove() επιστρέφει έναν ακέραιο διαφορετικό του μηδενός αν η λειτουργία αποτύχει.

Τα αρχεία stdin, stdout και stderr

'Ενα πρόγραμμα C παραλαμβάνει από το λειτουργικό σύστημα κάτω από το οποίο τρέχει τρία ανοικτά αρχεία που έχουν τα ονόματα *stdin*, *stdout* και *stderr*. Το αρχείο stdin είναι συνήθως συνδεμένο με το πληκτρολόγιο και ορίζει από που θα διαβάζουν τα δεδομένα οι συναρτήσεις εισόδου scanf(), gets(), κ.λ.π., ενώ το stdout είναι συνήθως συνδεμένο με την οθόνη του υπολογιστή μας και ορίζει το που θα τυπώνουν την έξοδό τους οι συναρτήσεις εξόδου, όπως η printf(), putchar() κ.ά. Το stderr, τέλος, είναι και αυτό συνήθως συνδεμένο με την οθόνη και χρησιμοποιείται για να εκτυπώνουμε μηνύματα λάθους (μέσω των συναρτήσεων fprintf(), fputc(), κ.ο.κ.) Λέμε "συνήθως" γιατί κάποιος, καλώντας το πρόγραμμα από το λειτουργικό σύστημα, έχει το δικαίωμα να αλλάξει τις συνδέσεις αυτές. Για παράδειγμα στο DOS και το Unix η είσοδος ενός προγράμματος αλλάζει καλώντας το με

```
program < file
```

και η έξοδος με

```
program > file
```

ή

```
program >> file
```

Το πρόγραμμά μας έχει το δικαίωμα να κλείσει τα αρχεία αυτά (χρησιμοποιώντας την fclose()), αλλά δεν μπορεί να τα ανοίξει χρησιμοποιώντας το συμβολισμό

```
stdin = fopen("datafile", "r");
```

Αν επιθυμούμε να αλλάξουμε τη σύνδεση ενός τέτοιου αρχείου, συνδέοντάς το με κάποιο συγκεκριμένο αρχείο, τότε θα χρησιμοποιήσουμε τη συνάρτηση *freopen()* της οποίας το πρότυπο είναι:

```
FILE *freopen(char *filename, char *mode, FILE *stream);
```

Η συνάρτηση αυτή κλείνει το αρχείο *stream* και κατόπιν ανοίγει το αρχείο *filename* με τρόπο προσπέλασης που καθορίζεται από το *mode* και συσχετίζει το ανοικτό

αρχείο με το *stream*. Η συνάρτηση επιστρέφει NULL αν η λειτουργία δεν επέτυχε. Αν θέλουμε λοιπόν οι συναρτήσεις εισόδου να παίρνουν την είσοδό τους από το αρχείο "myfile.dat" θα πρέπει να καλέσουμε την freopen() με

```
freopen("myfile.dat", "r", stdin);
```

Τυχαία προσπέλαση σε αρχεία

Από τη στιγμή που θα ανοίξουμε ένα αρχείο μπορούμε να διαβάσουμε σειριακά τα περιεχόμενά του με κλήσεις στις συναρτήσεις εισόδου από αρχεία, ή να γράψουμε σειριακά δεδομένα με αντίστοιχες κλήσεις σε συναρτήσεις. Αρκετές φορές όμως θέλουμε να διαβάσουμε ή να γράψουμε δεδομένα σε κάποια συγκεκριμένη θέση του αρχείου χωρίς να διαβάσουμε (ή να τροποποιήσουμε) τα προηγούμενα. Στην περίπτωση αυτή θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση *fseek()* της οποίας το πρότυπο είναι

```
int fseek(FILE *fp, long offset, int origin);
```

Η συνάρτηση αυτή έχει ως αποτέλεσμα να αλλάξει η θέση στην οποία θα γραφούν (ή από την οποία θα διαβαστούν) τα επόμενα δεδομένα όταν θα ξαναχρησιμοποιηθεί το αρχείο fp. Η νέα θέση καθορίζεται από τις παραμέτρους *offset* και *origin*. Το *offset* καθορίζει το πόσο θα μετακινηθεί ο δείκτης δεδομένων του αρχείου, και το *origin* ορίζει το σημείο από το οποίο θα μετρηθεί η μετακίνηση αυτή· αν το origin έχει τιμή SEEK_SET η απόσταση μετράται από την αρχή του αρχείου· αν έχει τιμή SEEK_CUR τότε η απόσταση μετράται από την τρέχουσα θέση· αν έχει την τιμή SEEK_END τότε η απόσταση μετράται από το τέλος του αρχείου (και πρέπει να είναι αρνητική ή μηδέν). Αν η συνάρτηση fseek() επιτύχει, τότε επιστρέφει την τιμή μηδέν.

Μπορούμε ακόμη να βρούμε και την τρέχουσα θέση του αρχείου χρησιμοποιώντας τη συνάρτηση ftell() η οποία δέχεται ως όρισμα έναν FILE * (που θα πρέπει να αντιστοιχεί σε ανοικτό αρχείο) και επιστρέφει έναν ακέραιο (long) που αντιστοιχεί στην τρέχουσα θέση του αρχείου (πόσοι χαρακτήρες υπάρχουν από την αρχή του αρχείου μέχρι την τρέχουσα θέση). Η ftell() επιστρέφει τη σταθερά -1L σε περίπτωση αποτυχίας.

Το πρόγραμμα του σχήματος 25 δημιουργεί ένα αρχείο με το όνομα "datafile" που περιέχει γραμμές της μορφής

```
ακέραιος1   ακέραιος2
```

Αρχικά το αρχείο περιέχει είκοσι γραμμές, στις οποίες ο $ακέραιος_1$ παίρνει τιμές από το 1 μέχρι το 20 και ο $ακέραιος_2$ είναι το διπλάσιο του $ακεραίου_1$. Το πρόγραμμα του σχήματος 26 αυξάνει τον $ακέραιο_2$ κατά ένα σε όλες τις γραμμές του αρχείου κάθε φορά που τρέχει, ενώ το πρόγραμμα του σχήματος 27 μας ζητάει έναν ακέραιο, έστω ι, και τυπώνει τη γραμμή υπ' αριθμόν ι του αρχείου, ή μας δίνει ένα διαγνωστικό μήνυμα αν η γραμμή αυτή δεν υπάρχει. Προσοχή στο ότι για κάθε ακέραιο χρησιμοποιείται καθορισμένος αριθμός ψηφίων για να μπορούμε να βρίσκουμε εύκολα τη θέση της ν-οστης γραμμής.

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

int main(int argc, char *argv[])
{
      FILE *fp;
      int i;

      if ((fp = fopen("datafile", "w")) == NULL)
      {
            fprintf(stderr, "Can't open datafile for output\n");
            exit(2);
```

```
	}

	for (i = 1; i <= 20; i++)
		fprintf(fp, "%11d\t%11d\n", i, 2 * i);
	fputc('\n', fp);
	fclose(fp);
	return 0;
}
```

Σχήμα 25

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

int main(int argc, char *argv[])
{
	FILE *fp;
	int item_read;

	if ((fp = fopen("datafile", "r+")) == NULL)
	{
		fprintf(stderr, "File datafile not found.\n");
		exit(2);
	}

	while (fscanf(fp, "%*d%d", &item_read) == 1)
	{
		fseek(fp, -11L, SEEK_CUR);
		fprintf(fp, "%11d", ++item_read);
		fseek(fp, 0L, SEEK_CUR);
	}
	fclose(fp);
	return 0;
}
```

Σχήμα 26

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
	FILE *fp;
	int line, i1, i2;

	printf("Enter line number to read: ");
	if (scanf("%d", &line) != 1)
	{
		fprintf(stderr, "Illegal user input.\n");
		exit(1);
	}

	if ((fp = fopen("datafile", "r")) == NULL)
	{
		fprintf(stderr,"Cannot open datafile for input \n");
		exit(2);
	}

	if ( (fseek(fp, 25 * line, SEEK_SET) != 0) ||
		scanf(fp, "%d%d", &i1, &i2) != 2)
	   )
	{
```

```
		fprintf(stderr, "Illegal line number.\n");
		exit(3);
	}
	fclose(fp);
	printf("Line %d holds ints %d and %d\n", line, i1, i2);
	return 0;
}
```

Σχήμα 27

Στο σχήμα 26 παρατηρούμε ότι υπάρχει μία κλήση της fseek() που εκ πρώτης όψεως δεν φαίνεται να έχει κανένα νόημα (fseek(fp, 0L, SEEK_CUR);). Η fseek() αυτή πραγματικά δεν μετακινεί το σημείο ανάγνωσης/εγγραφής του αρχείου, είναι όμως απαραίτητη γιατί δεν επιτρέπεται να έχουμε συνεχόμενες λειτουργίες ανάγνωσης/εγγραφής σε αρχείο χωρίς παρεμβολή της συνάρτησης fseek(). Έτσι σε ένα αρχείο που έχει ανοιχθεί ώστε να μπορούμε να διαβάζουμε και να γράφουμε δεδομένα πρέπει να κάνουμε συνεχόμενες λειτουργίες ανάγνωσης (εγγραφής) αλλά όταν χρειαστεί να κάνουμε κάποια λειτουργία εγγραφής (ανάγνωσης) θα πρέπει οπωσδήποτε να παρεμβάλουμε μία κλήση στη συνάρτηση fseek(), αλλιώς τα αποτελέσματα θα είναι απρόβλεπτα.