

Κεφάλαιο

4

Ο τύπος δεδομένων int



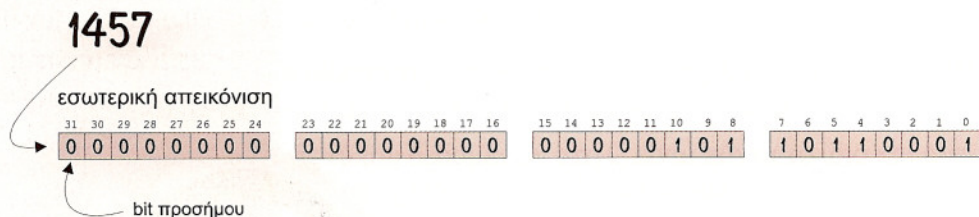
Ο τύπος δεδομένων int

Το κεφάλαιο αυτό εισάγει και αναλύει τον τρόπο χρήσης των ακέραιων (integer) στη γλώσσα C. Στο κεφάλαιο αυτό εξετάζεται η εσωτερική απεικόνιση των ακέραιων σταθερών και μεταβλητών, καθώς και οι τελεστές που εφαρμόζονται σε ακέραια μεγέθη. Πολλοί από αυτούς τους τελεστές εφαρμόζονται και σε άλλους τύπους δεδομένων.

Σταθερές τύπου int

Οι ακέραιες σταθερές δεν είναι τίποτε άλλο από απλοί ακέραιοι αριθμοί. Μια ακέραια σταθερά αποθηκεύεται στη μνήμη αφού μετατραπεί σε δυαδικό αριθμό. Μια σταθερά τύπου **int** μετατρέπεται σε ένα δυαδικό αριθμό μήκους 4 byte.

Με 4 byte μπορεί να αναπαρασταθεί αριθμός από -2.147.483.648 μέχρι +2.147.483.647. Το bit υπ' αριθμόν 31 (τα 32 bit αριθμούνται από το 0 έως το 31) χρησιμοποιείται για το πρόσημο και τα υπόλοιπα 31 για την αναπαράσταση του αριθμού.



Μεταβλητές τύπου int

Η C διαθέτει, πέρα από τον τύπο **int** που έχουμε ήδη αναφέρει, άλλους δύο τύπους ακέραιων μεταβλητών: **long int** και **unsigned int**.

Μια μεταβλητή τύπου **int** δηλώνεται ως εξής:

int όνομα.μεταβλητής;

Οι μεταβλητές τύπου `int` και `long int` δεσμεύουν 4 byte³ (32 bits) και μπορούν να αποθηκεύσουν έναν ακέραιο αριθμό από -2.147.483.648 μέχρι +2.147.483.647.

Μια μεταβλητή τύπου `long int` δηλώνεται ως εξής:

`long int όνομα.μεταβλητής;`

ή

`long όνομα.μεταβλητής;`

Στην πρόταση δήλωσης η λέξη `int` μπορεί να παραλειφθεί, αλλά για λόγους σαφήνειας αυτό δεν συνιστάται.

Μια μεταβλητή τύπου `unsigned int` δηλώνεται όπως παρακάτω:

`unsigned int ονομα.μεταβλητής;`

ή

`unsigned ονομα.μεταβλητής;`

Οι μεταβλητές τύπου `unsigned int` δεσμεύουν 4 byte (32 bit). Σε μια μεταβλητή τέτοιου τύπου, το bit υπ' αριθμόν 31 (το πρώτο bit του 32μπιτου αριθμού) δεν χρησιμοποιείται για το πρόσημο αλλά συμμετέχει στην απεικόνιση του αριθμού. Επομένως, σε μεταβλητές τύπου `unsigned int` μπορούν να αποθηκευτούν μόνο θετικοί αριθμοί.

Αρχική τιμή μεταβλητής

Όταν δηλώνουμε μια μεταβλητή μπορούμε ταυτόχρονα να της δώσουμε και μία αρχική τιμή. Για παράδειγμα, στην

```
int a=4,b=8;
```

γίνεται δήλωση δύο μεταβλητών `a` και `b`, στις οποίες ανατίθενται ταυτόχρονα οι τιμές 4 και 8 αντίστοιχα.

³ Σε παλαιότερα συστήματα 16bit, οι μεταβλητές τύπου `int` καταλαμβάνουν 2 byte ενώ οι `long int` 4.

Αριθμητικοί τελεστές

Εκτός από τους "κλασικούς" αριθμητικούς τελεστές, η C διαθέτει και μερικούς τελεστές οι οποίοι δεν συναντώνται σε άλλες γλώσσες προγραμματισμού. Ο διπλανός πίνακας αναφέρει τους διαθέσιμους αριθμητικούς τελεστές στη C.

Τελεστής	Λειτουργία
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο ακέραιας διαίρεσης
++	Αύξηση κατά 1
--	Μείωση κατά 1

Δεν θα ασχοληθούμε με τους κλασικούς αριθμητικούς τελεστές αλλά μόνο με τους νέους τελεστές ++ και -- που εισάγει η C.

Οι τελεστές ++ και -- εφαρμόζονται **μόνο** σε μεταβλητές και μπορεί να προηγούνται ή να ακολουθούν το όνομα μιας μεταβλητής.

π.χ. ++a; a++; --a; a--;

Ο τελεστής ++ αυξάνει το περιεχόμενο της μεταβλητής κατά 1, ενώ ο τελεστής -- μειώνει το περιεχόμενο της μεταβλητής κατά 1. Η λειτουργία αυτή γίνεται ανεξάρτητα από τη θέση του τελεστή (πριν ή μετά από τη μεταβλητή).

```
main()
{
    int a,b;
    b=a=5;
    ++a;
    --b;
    printf("a=%d b=%d\n",a,b);
}
```

Με την παράσταση ++a η a αυξάνεται κατά 1 (6) και με την παράσταση --b η b μειώνεται κατά 1 (4). Έτσι η printf() θα εμφανίσει στην οθόνη:
a=6 b=4

Όπως κάθε παράσταση, έτσι και οι παραστάσεις με τους τελεστές ++ και -- επιστρέφουν μία τιμή. Η τιμή αυτή εξαρτάται από τη θέση του τελεστή (πριν ή μετά από τη μεταβλητή).

☞ Όταν ο τελεστής είναι πριν από τη μεταβλητή, τότε πρώτα γίνεται η πράξη (αύξηση ή μείωση) και μετά επιστρέφεται η νέα τιμή της μεταβλητής (μετά από την αύξηση ή τη μείωση). Επομένως, στην παράσταση χρησιμοποιείται η νέα τιμή της μεταβλητής, μετά από την αύξηση ή τη μείωσή της. Για παράδειγμα, η **a=5; ++a;** αυξάνει το **a** κατά 1 και επιστρέφει την τιμή 6.

Όταν ο τελεστής είναι μετά τη μεταβλητή, τότε πρώτα επιστρέφει την τιμή της μεταβλητής (την παλιά τιμή, πριν από την πράξη) και μετά αυξάνει ή μειώνει το περιεχόμενο της μεταβλητής. Επομένως, στην παράσταση χρησιμοποιείται η παλιά τιμή της μεταβλητής πριν από την αύξηση της. Για παράδειγμα, η **a=5; a++;** επιστρέφει την τιμή 5 και αυξάνει το **a** κατά 1.

Στα επόμενα παραδείγματα γίνεται πιο σαφής η χρήση των τελεστών ++ και --.

Το επόμενο πρόγραμμα εμφανίζει τα αποτελέσματα που φαίνονται στο διπλανό πλαίσιο:

```
main()
{
    int a,b;
    a=b=5;
    printf("++a=%d\n", ++a);
    printf("b++=%d\n", b++);
    printf("a=%d b=%d\n", a,b);
}
```

```
++a=6
b++=5
a=6 b=6
```

Στην πρώτη **printf()** η παράσταση **++a** αυξάνει το περιεχόμενο της **a** κατά 1 (δηλαδή, γίνεται 6) και επιστρέφει την τιμή 6 (τιμή μετά την αύξηση) η οποία και εμφανίζεται.

Στη δεύτερη **printf()** η τιμή της παράστασης **b++** είναι 5 (τιμή πριν την αύξηση) οπότε θα εμφανιστεί το 5. Το περιεχόμενο της **b** όμως θα αυξηθεί κατά 1, δηλαδή θα γίνει 6.

Η τελευταία **printf()** εμφανίζει τα περιεχόμενα της **a** και **b**, δηλαδή 6 και 6 αντίστοιχα.

Τα αποτελέσματα του επόμενου προγράμματος εμφανίζονται στο διπλανό πλαίσιο:

```
main()
{
    int a,b,c,d;
    a=b=5;
    c=-b;
    d=a--;
    printf("a=%d b=%d c=%d d=%d\n",a,b,c,d);
}
```

a=4 b=4 c=4 d=5

Θα αναλύσουμε τις παραπάνω παραστάσεις με δεδομένο ότι (όπως θα δούμε αργότερα) ο τελεστής `--` έχει μεγαλύτερη προτεραιότητα από τον `=`. Στη παράσταση,

`c = --b;`

η φράση `--b` έχει αποτέλεσμα να μειωθεί η `b` κατά 1 και επιστρέφει ως αποτέλεσμα την τιμή της `b` μετά τη μείωση, δηλαδή το 4. Η τιμή αυτή καταχωρίζεται στην `c`. Στη παρακάτω παράσταση,

`d = a--;`

η φράση `a--` έχει αποτέλεσμα να μειωθεί η `a` κατά 1, αλλά επιστρέφει ως αποτέλεσμα την τιμή της `a` πριν από τη μείωση, δηλαδή το 5. Η τιμή αυτή καταχωρίζεται στην `d`.

Το αποτέλεσμα είναι ότι τελικά και η `a` και η `b` θα μειωθούν κατά 1 και θα έχουν την τιμή 4, αλλά ότι η `c` και η `d` θα πάρουν τις τιμές 4 και 5 αντίστοιχα.

Χρήση συντμήσεων σε παραστάσεις

Ο τελεστής ανάθεσης `=` μπορεί να συνδυαστεί με άλλους αριθμητικούς τελεστές για τη δημιουργία ορισμένων παραστάσεων σε συντετμημένη μορφή. Για παράδειγμα: η

Σύντμηση	Αντίστοιχη παράσταση
<code>x+=5</code>	<code>x=x+5</code>
<code>x-=5</code>	<code>x=x-5</code>
<code>x*=5</code>	<code>x=x*5</code>
<code>x/=5</code>	<code>x=x/5</code>
<code>x%=5</code>	<code>x=x%5</code>

$x = x + 5$ μπορεί να γραφεί και ως $x += 5$, ενώ η $x = x * 5$ μπορεί να γραφεί και ως $x *= 5$

Παρά το γεγονός ότι **δεν συνιστάται** η χρήση αυτών των συντμήσεων, είναι πιθανό να τις συναντήσουμε και αυτός είναι ο λόγος που τις αναφέρουμε. Ο παραπάνω πίνακας δείχνει με πέντε παραδείγματα τους συνδυασμούς σύντμησης και τις αντίστοιχες κανονικές παραστάσεις.

Ο τελεστής υπολοίπου %


Ο τελεστής % υπολογίζει το υπόλοιπο μιας ακέραιας διαίρεσης:

$10 \% 4 \rightarrow 2$

$5 \% 4 \rightarrow 1$

$5 \% 5 \rightarrow 0$

$4 \% 5 \rightarrow 4$

 Όταν ο διαιρέτης είναι μεγαλύτερος από τον διαιρετέο, το υπόλοιπο είναι ο διαιρετέος.

Δυαδικοί αριθμοί

Είναι γνωστό ότι για να επεξεργαστεί μια πληροφορία ο Η/Υ, τη μετατρέπει σε δυαδικό αριθμό. Η μετατροπή αυτή είναι συνήθως "αόρατη" από τον προγραμματιστή, ο οποίος δεν είναι απαραίτητο να γνωρίζει την εσωτερική απεικόνιση της πληροφορίας.

Η C δίνει τη δυνατότητα να χειριστούμε πληροφορίες σε επίπεδο δυαδικών αριθμών. Στη συνέχεια αυτού του κεφαλαίου θα γνωρίσουμε τελεστές οι οποίοι επιδρούν σε μεμονωμένα bit ενός ακέραιου αριθμού. Επιπρόσθετα, η γνώση της εσωτερικής απεικόνισης της πληροφορίας βοηθάει στην επεξήγηση και στην κατανόηση αρκετών λειτουργιών της C.

Μετατροπή δυαδικού σε δεκαδικό

Ένας δυαδικός αριθμός έχει ως βάση αρίθμησης το 2 (όπως ο δεκαδικός το 10). Κάθε δυαδικός αριθμός έχει έναν αντίστοιχο δεκαδικό και αντίστροφα. Τα

Συντελεστής	Δύναμη του 10	Σύνολο ψηφίου
4	$\times 1 (10^0)$	4
0	$\times 10 (10^1)$	0
2	$\times 100 (10^2)$	200
1	$\times 1000 (10^3)$	1000
Σύνολο		1204

ψηφία ενός αριθμού (δεκαδικού ή δυαδικού) αποτελούν τους συντελεστές των δυνάμεων της βάσης των (10 ή 2 αντίστοιχα).

Ας αναλύσουμε για παράδειγμα το δεκαδικό αριθμό 1204. Αρχίζοντας από τα δεξιά, το 4 είναι ο συντελεστής του 1 (10^0), το 0 του 10 (10^1), το 2 του 100 (10^2), και το 1 του 1000 (10^3). Επομένως ο αριθμός είναι

$$1 \cdot 1000 + 2 \cdot 100 + 0 \cdot 10 + 4 \cdot 1$$

δηλαδή 1204.

Αντίστοιχα τα ψηφία ενός δυαδικού αριθμού αποτελούν τους συντελεστές των δυνάμεων του 2 (και όχι του 10) δηλαδή 1, 2, 4, 8, 16, 32, 64, κ.ο.κ.

Ας αναλύσουμε για παράδειγμα το δυαδικό αριθμό $(1101)_2$. Αρχίζοντας από τα δεξιά, το 1 είναι ο συντελεστής του 1 (2^0), το 0 του 2 (2^1), το 1 του 4 (2^2), και το 1 του 8 (2^3). Επομένως ο αριθμός είναι:

Συντελεστής	Δύναμη του 2	Σύνολο ψηφίου
1	$\times 1 (2^0)$	1
0	$\times 2 (2^1)$	0
1	$\times 4 (2^2)$	4
1	$\times 8 (2^3)$	8
Σύνολο		13

$$1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$$

δηλαδή ο δεκαδικός 13.

Μετατροπή δεκαδικού σε δυαδικό

Κάθε δεκαδικός αριθμός έχει έναν αντίστοιχο δυαδικό. Ο τρόπος με τον οποίο βρίσκουμε τον αντίστοιχο δυαδικό αριθμό ενός δεκαδικού αναλύεται με το πα-

ρακάτω παράδειγμα, όπου βρίσκουμε τον αντίστοιχο δυαδικό αριθμό του 52. Θα ακολουθούμε τα επόμενα βήματα:

- $52/2 \rightarrow$ πηλίκο 26, υπόλοιπο 0 (το 0 αποτελεί τον συντελεστή του 2^0)
 $26/2 \rightarrow$ πηλίκο 13, υπόλοιπο 0 (το 0 αποτελεί τον συντελεστή του 2^1)
 $13/2 \rightarrow$ πηλίκο 6, υπόλοιπο 1 (το 1 αποτελεί τον συντελεστή του 2^2)
 $6/2 \rightarrow$ πηλίκο 3, υπόλοιπο 0 (το 0 αποτελεί τον συντελεστή του 2^3)
 $3/2 \rightarrow$ πηλίκο 1, υπόλοιπο 1 (το 1 αποτελεί τον συντελεστή του 2^4)
 $1/2 \rightarrow$ πηλίκο 0, υπόλοιπο 1 (το 1 αποτελεί τον συντελεστή του 2^5)

Το γεγονός ότι στην τελευταία διαίρεση το πηλίκο είναι 0, σηματοδοτεί το τέλος της διαδικασίας. Επομένως ο αντίστοιχος δυαδικός αριθμός του 52 είναι ο $(110100)_2$. Πραγματικά, αν κάνουμε τις πράξεις:

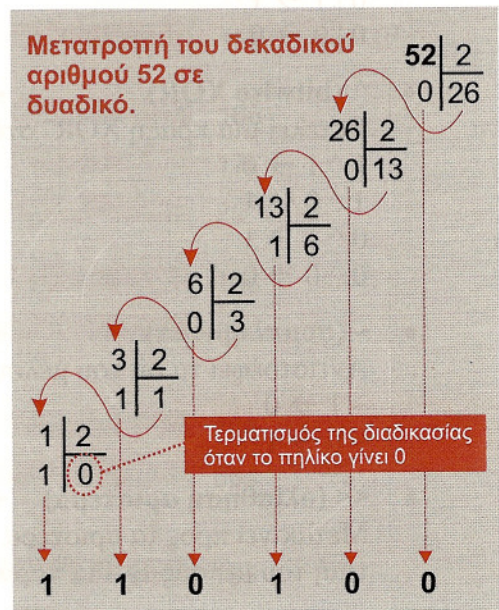
$$1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \rightarrow 32 + 16 + 0 + 4 + 0 + 0 \rightarrow 52$$

Η διαδικασία μετατροπής του αριθμού 52 στον αντίστοιχο δυαδικό αριθμό φαίνεται στο σχήμα.

Διαιρούμε το 52 με το 2. Έχει υπόλοιπο 0 και πηλίκο 26. Το υπόλοιπο αποτελεί το τελευταίο ψηφίο του δυαδικού αριθμού

Διαιρούμε το πηλίκο που βρήκαμε (26) πάλι με το 2. Το νέο υπόλοιπο αποτελεί το προτελευταίο ψηφίο του δυαδικού αριθμού. Αυτό συνεχίζεται μέχρι το πηλίκο να γίνει 0.

Το τελευταίο υπόλοιπο αποτελεί το πρώτο ψηφίο του δυαδικού αριθμού. Έτσι, το **52** μετατρέπεται στον δυαδικό $(110100)_2$.



Τελεστές bitwise

Οι τελεστές αυτοί επιτρέπουν στη C να χειρίζεται τους ακέραιους αριθμούς σε επίπεδο bit (bitwise="κατά bit"). Όλοι οι τελεστές bitwise, εκτός από τον τελεστή \sim (συμπλήρωμα), έχουν δύο μέλη, ένα αριστερό και ένα δεξιό. Τα μέλη αυτά (μέλος-A, μέλος-B) πρέπει να είναι ακέραιες παραστάσεις, δηλαδή να αποδίδουν ακέραιες τιμές. Οι τελεστές bitwise είναι έξι:

- **& (bitwise AND)** \rightarrow μέλος-A & μέλος-B
Εκτελεί μια πράξη AND στα αντίστοιχα bit των δύο μελών.
 $1 \& 1 \rightarrow 1$
 $1 \& 0 \rightarrow 0$
 $0 \& 1 \rightarrow 0$
 $0 \& 0 \rightarrow 0$
- **| (bitwise OR)** \rightarrow μέλος-A | μέλος-B
Εκτελεί μια πράξη OR στα αντίστοιχα bit των δύο μελών.
 $1 | 1 \rightarrow 1$
 $1 | 0 \rightarrow 1$
 $0 | 1 \rightarrow 1$
 $0 | 0 \rightarrow 0$
- **^ (bitwise XOR)** \rightarrow μέλος-A ^ μέλος-B
Εκτελεί μια πράξη XOR⁴ στα αντίστοιχα bit των δύο μελών.
 $1 \wedge 1 \rightarrow 0$
 $1 \wedge 0 \rightarrow 1$
 $0 \wedge 1 \rightarrow 1$
 $0 \wedge 0 \rightarrow 0$
- **\sim (συμπλήρωμα)** $\rightarrow \sim$ μέλος-A
Αντιστρέφει τα bit του μέλους-A
 $\sim 1 \rightarrow 0$
 $\sim 0 \rightarrow 1$
- **<< (ολίσθηση αριστερά)** \rightarrow μέλος-A << μέλος-B
Μετακινεί προς τα αριστερά όλα τα bit του μέλους-A τόσες θέσεις όση η τιμή του μέλους-B. Για παράδειγμα, η

⁴ Το λογικό XOR είναι αληθές (1) όταν μόνο ένα από τα δύο μέλη του είναι αληθές (1). Στην περίπτωση που και τα δύο είναι αληθή, τότε το αποτέλεσμα είναι ψευδές (0).

$a \ll 3$ μετακινεί τα bit της μεταβλητής a τρεις θέσεις αριστερά.

- \gg (ολίσθηση δεξιά) \rightarrow μέλος-A \gg μέλος-B

Μετακινεί προς τα δεξιά όλα τα bit του μέλους-A τόσες θέσεις όση η τιμή του μέλους-B. Για παράδειγμα, η

$a \gg 3$ μετακινεί τα bit της μεταβλητής a τρεις θέσεις δεξιά.

Στα επόμενα παραδείγματα, φαίνεται ο τρόπος με τον οποίο οι τελεστές bitwise επιδρούν στα bit των μελών τους.

$\&$ (bitwise AND)

$166 \& 176 \rightarrow 160$

$\&$

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

166

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

176

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

160

$|$ (bitwise OR)

$166 | 176 \rightarrow 182$

$|$

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

166

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

176

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

182

\wedge (bitwise XOR)

$166 \wedge 176 \rightarrow 22$

\wedge

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

166

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

176

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

22

\sim (συμπλήρωμα)

$\sim 166 \rightarrow 89$

\sim

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

166

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

89

\ll (ολίσθηση αριστερά)

$166 \ll 2 \rightarrow 664$

\ll

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

166

15	14	13	12	11	10	9	8
0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0

664

\gg (ολίσθηση δεξιά)

$166 \gg 2 \rightarrow 41$

\gg

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

166

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

41



Δεν πρέπει να γίνεται σύγχυση των συνδυαστικών λογικών τελεστών (&&, ||) με τους τελεστές bitwise (&, |). Ας θεωρήσουμε το επόμενο παράδειγμα:

```
main()
{
    int a,b,c,d;
    a=166;
    b=176;
    c=a && b;
    d=a & b;
    printf("c=%d d=%d\n",c,d);
}
```

c=1 d=160

Η λογική παράσταση $a \ \&\& \ b$ έχει αποτέλεσμα 1 (αληθές) επειδή τα μέλη a και b θεωρούνται αληθή δεδομένου ότι αποδίδουν τιμές διάφορες του 0 (166 και 176 αντίστοιχα).

Αντίθετα η παράσταση $a \ \& \ b$ επιβάλλει την πράξη bitwise AND μεταξύ του 166 και του 176 με αποτέλεσμα 160.

```
main()
{
    int a,b,c,d;
    a=8;
    b=4;
    c=a || b;
    d=a | b;
    printf("c=%d d=%d\n",c,d);
}
```

c=1 d=12


Η λογική παράσταση $a \ || \ b$ έχει αποτέλεσμα 1 (αληθές), ενώ η παράσταση $a \ | \ b$ επιβάλλει την πράξη bitwise OR μεταξύ του 8 και του 4, με αποτέλεσμα 12.


Προτεραιότητα τελεστών

Στη C μπορεί μέσα σε μια παράσταση να συνυπάρχουν αριθμητικοί και λογικοί τελεστές, δεδομένου ότι τελικά όλοι αποδίδουν αριθμητικές τιμές.

Ο διπλάνος πίνακας δείχνει την προτεραιότητα των τελεστών αλλά και τη φορά εκτέλεσής τους στην περίπτωση που υπάρχουν πολλοί με την ίδια προτεραιότητα. Οι τελεστές ανάμεσα σε δύο έντονες οριζόντιες γραμμές έχουν την ίδια προτεραιότητα. Π.χ. ο πολλαπλασιασμός, η διαίρεση, και το υπόλοιπο (%) έχουν την ίδια προτεραιότητα.

Η παράσταση $10/2*5$ θα έχει αποτέλεσμα 25. Οι δύο τελεστές (/ και *) έχουν την ίδια προτεραιότητα και σύμφωνα με τον διπλάνο πίνακα η φορά εκτέλεσης των δύο πράξεων θα είναι από τα αριστερά προς τα δεξιά (\Rightarrow). Πρώτα λοιπόν θα γίνει η διαίρεση ($10/2$) και μετά ο πολλαπλασιασμός ($5*5$).

 Για να αλλάξει η σειρά εκτέλεσης των πράξεων μπορούν να χρησιμοποιηθούν παρενθέσεις (). Η χρήση τους επιβάλλεται επίσης στα σημεία για τα οποία δεν είμαστε σίγουροι για την προτεραιότητα των τελεστών. Π.χ η παράσταση $10/(2*5)$ θα έχει αποτέλεσμα 1 διότι θα γίνει πρώτα η πράξη μέσα στις παρενθέσεις.

 Παρατηρούμε ότι ο τελεστής ανάθεσης, το ίσον (=), έχει τη μικρότερη προτεραιότητα.

Προτεραιότητα τελεστών		
Τελεστής	Σημασία	Φορά
!	Λογικό NOT	\Leftarrow
~	Συμπλήρωμα	\Leftarrow
++	Αύξηση	\Leftarrow
--	Μείωση	\Leftarrow
*	Πολ/σμός	\Rightarrow
/	Διαίρεση	\Rightarrow
%	Υπόλοιπο	\Rightarrow
+	Πρόσθεση	\Rightarrow
-	Αφαίρεση	\Rightarrow
<<	Ολίσθηση αριστερά	\Rightarrow
>>	Ολίσθηση δεξιά	\Rightarrow
<	Μικρότερο	\Rightarrow
<=	Μικρότερο ή ίσο	\Rightarrow
>	Μεγαλύτερο	\Rightarrow
>=	Μεγαλύτερο ή ίσο	\Rightarrow
==	Ίσο	\Rightarrow
!=	Όχι ίσο	\Rightarrow
&	Bitwise AND	\Rightarrow
^	Bitwise XOR	\Rightarrow
	Bitwise OR	\Rightarrow
&&	Λογικό AND	\Rightarrow
	Λογικό OR	\Rightarrow
?:	OR συνθήκης	\Leftarrow
=	Ανάθεση	\Leftarrow

Στο διπλάνο πίνακα υπάρχουν και τελεστές στους οποίους θα αναφερθούμε αργότερα. Προς το παρόν θα χρησιμοποιήσουμε τους πιο απαραίτητους τελεστές.

Αναφέραμε προηγουμένως ότι στη C οι λογικές παραστάσεις επιστρέφουν αριθμητικές τιμές: 1 για αλήθεια και 0 για ψέμα. Επομένως δεν πρέπει να μας παραξενεύουν οι παραστάσεις της μορφής:

`a=b>c;`

Σύμφωνα με τον πίνακα προτεραιότητας, πρώτα θα υπολογιστεί η παράσταση `b>c`. Επομένως στην `a` θα καταχωριστεί το 1 αν η `b` είναι μεγαλύτερη από τη `c`, διαφορετικά θα καταχωριστεί το 0.

`a=b==c;`

Στην `a` θα καταχωριστεί το 1 αν η `b` είναι ίση με τη `c`, διαφορετικά θα καταχωριστεί το 0.

Παραδείγματα

Π.1 Στις επόμενες τέσσερις προτάσεις υποθέτουμε ότι το `x` και το `y` έχουν τιμή 100 και 4 αντίστοιχα πριν από την εκτέλεση **κάθε** παράστασης. Συμπληρώστε την τιμή του `x`, την τιμή του `y`, και την τιμή της παράστασης μετά από την εκτέλεση της κάθε πρότασης.

Πρόταση	Τιμή του x	Τιμή του y	Τιμή της παράστασης
<code>y * x++;</code>	101	4	400 (4 * 100)
<code>++x * --y;</code>	101	3	303 (101 * 3)
<code>x-- + ++y;</code>	99	5	105 (100 + 5)
<code>--x * 2;</code>	99	4	198 (99 * 2)
<code>x=x>y++</code>	1	5	1
<code>(x<y) (y>5)</code>	100	4	0
<code>++x>100</code>	101	4	1
<code>x++>100</code>	101	4	0

Π.2 Υποθέτουμε ότι μία μετεωρολογική συσκευή μέτρησης, συνδεδεμένη με τον Η/Υ επιστρέφει την κατάσταση της και τη μέτρηση σε ένα αριθμό ως εξής:

Το bit 7 του αριθμού έχει τιμή 1 αν η συσκευή λειτουργεί σωστά. Τα bit 6 και 5 αναφέρουν το είδος της μέτρησης (0-Θερμοκρασία, 1-Πίεση, 2-Υγρασία, 3-Ταχύτητα ανέμου). Τα υπόλοιπα bit (4, 3, 2, 1, και 0) αποτελούν την τιμή της μέτρησης.

OK	Είδος	Είδος	Τιμή	Τιμή	Τιμή	Τιμή	Τιμή
7	6	5	4	3	2	1	0

Ο επόμενος κώδικας αναλύει τα bit του αριθμού και εμφανίζει την κατάσταση της συσκευής, το είδος της μέτρησης, και την τιμή της:

```
main()
{
    int ar;
    int ok,eidos,timi;
    printf("Δώσε αριθμό από τη συσκευή:");
    scanf("%d",&ar);
    ok=ar&128>>7;
    ειδος=(ar&96)>>5;
    τιμι=ar&31;
    printf("OK=%d Είδος=%d Τιμή=%d\n",ok,eidos,τιμι);
}
```

Ο τελεστής & μαζί με κάποια σταθερή τιμή, χρησιμοποιείται για να απομονώσει συγκεκριμένα bit του αριθμού. Π.χ. η `ar&96` απομονώνει τα bit 6 και 5 του `ar`. Η εφαρμογή του τελεστή `>>5` ολισθαίνει τα 2 bit 5 θέσεις δεξιά ώστε να είναι στην αρχή του byte και να αποδώσουν τιμές από 0 μέχρι 3.

96							
0	1	1	0	0	0	0	0

Η παράσταση `ar&128` απομονώνει το bit 7 του αριθμού και η εφαρμογή του τελεστή `>>7` ολισθαίνει το bit 7 θέσεις δεξιά ώστε να είναι στην αρχή του byte και να αποδώσει τιμή 0 ή 1.

128							
1	0	0	0	0	0	0	0

Ανάλογα απομονώνονται και τα υπόλοιπα bit που ενδιαφέρουν.

Ανασκόπηση Κεφαλαίου 4

- Ο τύπος δεδομένων `int` αφορά ακέραιους αριθμούς. Οι μεταβλητές τύπου `int` δεσμεύουν 4 byte.
- Κατά τη δήλωσή της, μία μεταβλητή μπορεί να πάρει ταυτόχρονα και αρχική τιμή, π.χ. `int a=5;`
- Ο τελεστής `++` αυξάνει μία μεταβλητή τύπου `int` κατά ένα, ενώ ο τελεστής `--` τη μειώνει κατά 1.
- Όταν ο τελεστής (`++` ή `--`) είναι πριν τη μεταβλητή, τότε πρώτα γίνεται η πράξη (αύξηση ή μείωση) και μετά επιστρέφεται η νέα τιμή της μεταβλητής (μετά από την αύξηση ή μείωση).
- Όταν ο τελεστής (`++` ή `--`) είναι μετά τη μεταβλητή, τότε πρώτα επιστρέφεται η τιμή της μεταβλητής (η παλιά τιμή, πριν από την πράξη) και μετά αυξάνεται ή μειώνεται το περιεχόμενο της μεταβλητής.

Ασκήσεις Κεφαλαίου 4

- 4.1** Στις επόμενες τέσσερις προτάσεις υποθέτουμε ότι το `x` έχει τιμή 100 πριν από την εκτέλεση **κάθε** παράστασης. Συμπληρώστε την τιμή του `x` και την τιμή της παράστασης μετά από την εκτέλεση της κάθε πρότασης. ★

Πρόταση	Τιμή του x	Τιμή της παράστασης
<code>x++;</code>		
<code>++x;</code>		
<code>x--;</code>		
<code>--x;</code>		

- 4.2** Τι αποτέλεσμα θα έχει το επόμενο πρόγραμμα; ★★

```
main()
{
    int a,b,aa,bb,x,y;
    x = y = 100;
    a = ++x;
```

```

b = y++;
aa = ++x;
bb = y++;
printf("Η τιμή του a είναι %d\n",a);
printf("Η τιμή του b είναι %d\n",b);
printf("Η τιμή του aa είναι %d\n",aa);
printf("Η τιμή του bb είναι %d\n",bb);
}

```

- 4.3** Υποθέτουμε ότι η τιμή του **y** είναι 100 πριν από την εκτέλεση **κάθε** μίας από τις επόμενες παραστάσεις. Ποιες θα είναι οι τιμές των μεταβλητών **x** και **y** μετά από την εκτέλεση κάθε παράστασης; ★★

Παράσταση	Τιμή του x	Τιμή του y
x=y;		
x = -y * 4;		
x = y = y++;		
x = y == 100;		
x = y == y++;		
x = y == ++y;		

- 4.4** Δεδομένου του επόμενου τμήματος κώδικα:

```

int x,y,z;
x = 10;
y = 3;
z = (x / y) * y;

```

συμπληρώστε το κατάλληλα ώστε να συγκρίνει τα **x** και **z** και να τυπώνει ανάλογα: ★★

x == z αν το **x** είναι ίσο με το **z**
x < z αν το **x** είναι μικρότερο από το **z**
x > z αν το **x** είναι μεγαλύτερο από το **z**

- 4.5 Να γραφούν οι επόμενες τρεις προτάσεις **σαν μία πρόταση**, χρησιμοποιώντας τον τελεστή ++. ★ ★

```
y = y + 1;
```

```
z = x + y;
```

```
x = x + 1;
```

- 4.6 Ποιοι είναι οι αντίστοιχοι δυαδικοί αριθμοί των επόμενων δεκαδικών ακεραίων: ★ ★

15

52

0

128

- 4.7 Ποιοι είναι οι αντίστοιχοι δεκαδικοί αριθμοί των επόμενων δυαδικών: ★ ★

1100111

111

1000000

- 4.8 Τι αποτέλεσμα θα έχει το επόμενο πρόγραμμα; ★ ★

```
main()
```

```
{
```

```
    int a,b,c;
```

```
    a=5;
```

```
    b=8;
```

```
    printf("%d \n%d\n %d\n",a & b, a | b, a && b);
```

```
}
```

4.9 Ποια από τα επόμενα αληθεύουν: ★

- ☐ Το `i++` αυξάνει την τιμή του `i` κατά 1 ενώ το `++i` όχι.
- ☐ Οι τελεστές `++` και `--` εφαρμόζονται **μόνο** σε μεταβλητές.
- ☐ Όταν κάνω μια πράξη bitwise AND (`&`) με το 0, το αποτέλεσμα θα είναι πάντα 0.
- ☐ Ο τελεστής ανάθεσης = έχει την πρώτη προτεραιότητα.
- ☐ Η παράσταση `5/2` έχει αποτέλεσμα τύπου `int` (το 2).

4.10 Υποθέτουμε ότι η τιμή του `x` είναι 5, του `y` είναι 100, και του `a` είναι 0, πριν από την εκτέλεση **κάθε** μίας από τις επόμενες παραστάσεις. Ποιες θα είναι οι τιμές των μεταβλητών `x` και `y` μετά από την εκτέλεση κάθε παράστασης; ★★

Παράσταση	Τιμή του x	Τιμή του y
<code>x = y > x a</code>		
<code>x = y a;</code>		
<code>y = x & a;</code>		
<code>x = x & y;</code>		
<code>x = x y;</code>		
<code>x = --x && y a;</code>		