2. Μεταβλητές – Είσοδος/έξοδος προγράμματος – Εκφράσεις και Τελεστές

Σύνοψη
Στο κεφάλαιο αυτό ο αναγνώστης εισάγεται στην έννοια και τη χρήση της μεταβλητής στον προγραμματισμό. Αναλύονται τα είδη μεταβλητών στη γλώσσα C, η εσωτερική απεικόνισή τους και τα λειτουργικά χαρακτηριστικά τους. Ακολούθως, περιγράφονται οι βασικές συναρτήσεις ανάγνωσης και εκτύπωσης δεδομένων της γλώσσας C, οι οποίες επιτελούν την επικοινωνία του προγράμματος με το εξωτερικό περιβάλλον (συσκευές εισόδου και εξόδου). Στη συνέχεια, παρουσιάζονται οι κατηγορίες εκφράσεων και μελετώνται τα είδη των τελεστών: αύξησης– μείωσης, ανάθεσης, συσχετιστικοί και λογικοί τελεστές, καθώς και ο τελεστής sizeof. Ακολουθεί η ανάλυση των εννοιών και των ιδιοτήτων της προτεραιότητας και της προσεταιριστικότητας και το κεφάλαιο ολοκληρώνεται με τις διαδικασίες της ρητής και της έμμεσης μετατροπής τύπων.

Λέξεις κλειδιά
μεταβλητή, σταθερά, τύπος χαρακτήρα, τύπος ακεραίου, τύπος αριθμού κινητής υποδιαστολής, εκφράσεις, σημειολογίες τελεστών, αριθμητικοί τελεστές, λογικοί τελεστές, συσχετιστικοί τελεστές, τελεστές αύξησης– μείωσης, προτεραιότητα και προσεταιριστικότητα τελεστών, ρητή– έμμεση μετατροπή τύπου, scanf, printf, puchar, getchar, sizeof

Προαπαιτούμενη γνώση
Λεξιλόγιο της γλώσσας C– στάδια υλοποίησης προγράμματος

2.1. Η έννοια της μεταβλητής

Ένα από τα πλέον βασικά πλεονεκτήματα του ηλεκτρονικού υπολογιστή είναι η δυνατότητά του να διαχειρίζεται πληροφορία σε μορφή αριθμητικών δεδομένων, γραμμάτων ή ακολουθίας γραμμάτων. Τα δεδομένα αποθηκεύονται στη μνήμη και θα πρέπει να μπορούν να είναι προσβάσιμα από τα προγράμματα, να εισαχθούν σε υπολογισμούς και να δημιουργήσουν νέα δεδομένα. Αρχικά, οι προγραμματιστές χειρίζονταν τα δεδομένα δουλεύοντας με τις διευθύνσεις μνήμης, στις οποίες ήταν αποθηκευμένα. Με την αύξηση του μεγέθους και της πολυπλοκότητας των προγραμμάτων, αυτός ο τρόπος διαχείρισης έθετε πολλούς περιορισμούς και αποτελούσε πηγή προβλημάτων.

Η λύση στο πρόβλημα της αναφοράς σε δεδομένα και στη διαχείρισή τους δόθηκε με την εισαγωγή της έννοιας των μεταβλητών, οι οποίες διαχειρίζονται δεδομένα. Το όνομα μίας μεταβλητής είναι άμεσα συνδεδεμένο με τη διεύθυνση στην οποία είναι αποθηκευμένο το δεδομένο. Με τον τρόπο αυτό ο προγραμματιστής μπορεί να χειριστεί δεδομένα χωρίς να γνωρίζει την ακριβή διεύθυνση της μνήμης όπου αυτά τοποθετούνται.

Επιγραμματικά, η χρήση των μεταβλητών είναι ίδια με εκείνη της άλγεβρας, π.χ. η παράσταση `y=3*x+5` ισχύει τόσο στα μαθηματικά όσο και στις γλώσσες προγραμματισμού, με τα `x` και `y` να είναι μεταβλητές. Ωστόσο, στον προγραμματισμό η χρήση της είναι γενικευμένη: **η μεταβλητή είναι μία θέση μνήμης για ένα δεδομένο**. Δημιουργείται, όταν δηλώνεται, και η τιμή της δεν είναι καθορισμένη, έως ότου χρησιμοποιηθεί για πρώτη φορά από το πρόγραμμα. Η γλώσσα C απαιτεί δήλωση μίας μεταβλητής πριν τη χρήση της, καθότι η δήλωση συνεπάγεται τον προσδιορισμό του τύπου δεδομένου που θα χειρίζεται και, συνακόλουθα, τη μνήμη που θα πρέπει να της ανατεθεί. Συνήθως οι μεταβλητές δηλώνονται στην αρχή της

συνάρτησης, ως επί το πλείστον αμέσως μετά το εισαγωγικό άγκιστρο (`{`). Ωστόσο, το πρότυπο της γλώσσας C99 επιτρέπει τη δήλωση σε οποιοδήποτε σημείο του κώδικα, πάντοτε όμως πριν την πρώτη χρήση τους.

2.1.1. Δήλωση μεταβλητής

Οι μεταβλητές δηλώνονται με ***πρόταση ορισμού***, η οποία τελειώνει πάντοτε με (`;`). Η μορφή της δήλωσης είναι: `data_type var, var, … ;`

π.χ. `int counter1, counter2;`

Οι μεταβλητές δηλώνονται στην αρχή μίας συνάρτησης. Η δήλωση γνωστοποιεί στον μεταγλωττιστή το όνομα της μεταβλητής και τον τύπο δεδομένου που θα χειρίζεται. Μία δήλωση έχει ως αποτέλεσμα τη σύνδεση του ονόματος της μεταβλητής με:

- τον ανάλογο τύπο δεδομένων, γεγονός που λαμβάνει χώρα στον χρόνο μεταγλώττισης (compile–time),
- μία θέση μνήμης κατάλληλου μεγέθους, γεγονός που λαμβάνει χώρα στον χρόνο εκτέλεσης (run–time).

2.1.2. Ονομασία μεταβλητής

Σε ό,τι αφορά την ονοματολογία, ακολουθούνται οι εξής κανόνες:

- Στη γλώσσα C τα ονόματα των μεταβλητών σχηματίζονται από:
  (α) τα γράμματα του αλφαβήτου,
  (β) τα ψηφία 0 έως 9,
  (γ) τον χαρακτήρα υπογράμμισης **_** (underscore).
- Το όνομα πρέπει να ξεκινά με γράμμα ή με χαρακτήρα υπογράμμισης (στη δεύτερη περίπτωση ο επόμενος χαρακτήρας πρέπει να είναι μικρό γράμμα).
- Το όνομα δεν πρέπει να είναι ίδιο με δεσμευμένη λέξη.
- Τα όνομα μίας μεταβλητής πρέπει να είναι ενδεικτικό της ιδιότητάς της ή του τύπου δεδομένου που αντιπροσωπεύει, έτσι ώστε να δίνει πληροφορία στον προγραμματιστή. Στην περίπτωση που το όνομα μίας μεταβλητής περιέχει περισσότερες από μία λέξεις, συνιστάται το πρώτο γράμμα κάθε λέξης να είναι κεφαλαίο.

Με βάση τα παραπάνω, ακολουθούν ενδεικτικές περιπτώσεις ονοματολογίας:

- Έγκυρα ονόματα μεταβλητών:
  `totalArea  max_amount  counter1`
  `Counter1  _temp_in_F`
- Μη έγκυρα ονόματα μεταβλητών:
  `$product  total%  3rd`
- Απαράδεκτα ονόματα μεταβλητών:
  `j  x2  max_number_of_students_in_my_class`

Σημείωση: Στην πορεία ανάγνωσης του κειμένου ο αναγνώστης θα παρατηρήσει ότι οι ονομασίες των μεταβλητών δε συνάδουν πάντοτε με τους κανόνες που αναφέρονται, καθώς χρησιμοποιούνται μεταβλητές με ένα ή δύο γράμματα. Η χρήση τους γίνεται *συγγραφική αδεία*, επιβληθείσα για λόγους πρακτικούς, έτσι ώστε να μη διευρύνεται ο κώδικας.

2.2. Τύποι μεταβλητών

Οι μεταβλητές της γλώσσας C ανήκουν σε δύο κατηγορίες. Η κατηγορία των βαθμωτών τύπων περιλαμβάνει:

- Τους ακέραιους (integers), οι οποίοι δηλώνονται με την κωδική λέξη `int`.
- Τους πραγματικούς, οι οποίοι χωρίζονται στους αριθμούς κινητής υποδιαστολής με κωδική λέξη `float` και τους αριθμούς διπλής ακρίβειας με κωδική λέξη `double`.

- Τη μεταβλητή χαρακτήρα (character, **char**).
- Τους δείκτες (pointers).
- Τον απαριθμητικό τύπο (enumerated, **enum**).

Στο πρότυπο της γλώσσας C99 προστέθηκε και ο τύπος **bool**, ο οποίος εμφανίζει δύο τιμές (**true** που αντιστοιχεί στον ακέραιο **1** και **false** που αντιστοιχεί στον ακέραιο **0**). Επειδή όμως στην πραγματικότητα αυτός ο τύπος είναι τεχνητός, υπό την έννοια ότι τα αποτελέσματα προκύπτουν μέσα από επεξεργασία ακεραίων από μακροεντολές, δεν θα μελετηθεί.

Στην κατηγορία των συναθροιστικών τύπων ανήκουν οι πίνακες, οι δομές (**struct**) και οι ενώσεις (**union**). Στη συνέχεια του κεφαλαίου γίνεται αναφορά στους βασικούς τύπους της C: **char**, **int**, **float**, **double** και στο Κεφάλαιο 5 αναπτύσσονται οι πίνακες. Οι άλλοι τύποι μεταβλητών θα μελετηθούν στο Κεφάλαιο 8.

2.2.1. Ο τύπος του χαρακτήρα

Ο τύπος του χαρακτήρα (**char**) παριστάνει χαρακτήρες του αλφάβητου της γλώσσας. Βρίσκεται ανάμεσα σε εισαγωγικά (π.χ. **'C'**, **'2'**, **'*'**, **')'**).

Η δήλωση της μεταβλητής χαρακτήρα ακολουθεί τον εξής φορμαλισμό:

```
char <όνομα_μεταβλητής>;   π.χ.   char choice;
```

Υπάρχει η δυνατότητα ταυτόχρονα με τη δήλωση να αποδοθεί αρχική τιμή στη μεταβλητή:

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

Η εισαγωγή τιμών στις μεταβλητές χαρακτήρα γίνεται με χρήση της συνάρτησης **scanf()** και του **προσδιοριστή** **%c** (character). Η πρόταση

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

διαβάζει από την κύρια είσοδο (πληκτρολόγιο) έναν χαρακτήρα και τον αποδίδει στη μεταβλητή **choice**. Θα πρέπει να προσεχθεί η χρήση του **&** πριν από τη μεταβλητή. Ονομάζεται **τελεστής διεύθυνσης** και προηγείται των μεταβλητών στη συνάρτηση **scanf()** (με εξαίρεση τα ονόματα μεταβλητών που αφορούν σε πίνακες, για τους οποίους θα γίνει σχετικός σχολιασμός στο Κεφάλαιο 5).

Η εκτύπωση μεταβλητών χαρακτήρα γίνεται με χρήση της συνάρτησης **printf()** και του προσδιοριστή **%c**. Η πρόταση

```
printf( "The character is %c\n", choice );
```

θα τυπώσει (θεωρώντας ότι στη **choice** εισήχθη ο **'A'**)

```
The character is A
```

Παρατήρηση: Επειδή κάθε χαρακτήρας του κώδικα ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) αντιστοιχεί σε έναν οκταψήφιο δυαδικό αριθμό, εάν αντί του **%c** χρησιμοποιηθεί ο προσδιοριστής **%d** (decimal), η **printf()** θα εμφανίσει τον ASCII κωδικό τού χαρακτήρα. Η πρόταση

```
printf( "The ASCII Code of %c is %d\n", choice,choice);
```

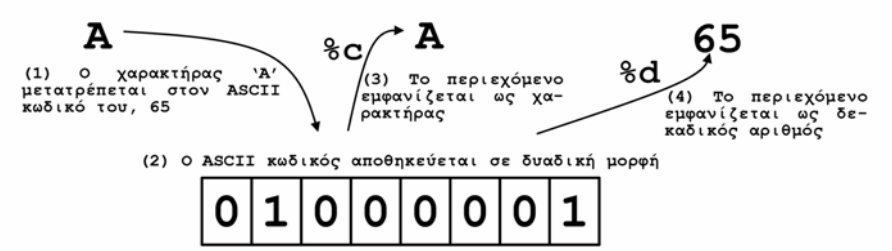
θα τυπώσει

```
The ASCII Code of A is 65
```

όπου **65** είναι το δεκαδικό ισοδύναμο του δυαδικού κωδικού ASCII για τον χαρακτήρα **'A'**.

Ο μεταγλωττιστής απαιτεί 1 byte μνήμης για την αποθήκευση της τιμής μίας μεταβλητής χαρακτήρα. Στο **Σχήμα 2.1** παρουσιάζονται οι διαδικασίες της αποθήκευσης και ανάκλησης για τον χαρακτήρα **A**. Η τιμή της μεταβλητής μετατρέπεται σε ακέραιο (ενέργεια 1 του **Σχήματος 2.1**), ο οποίος αποθηκεύεται (ενέργεια 2). Στην περίπτωση ανάκλησης της τιμής, εκτελείται η αντίστροφη διεργασία. Ο αριθμός μετατρέπεται σε χαρακτήρα μέσω του προσδιοριστή **%c** και ακολούθως είτε τυπώνεται ο χαρακτήρας (ενέργεια 3) είτε

τυπώνεται το δεκαδικό ισοδύναμο μέσω του προσδιοριστή `%d` (ενέργεια 4). Σε κάθε περίπτωση, ο μεταγλωττιστής είναι υπεύθυνος για το ότι ο υπολογιστής διαχειρίζεται τα bits και bytes σύμφωνα με τους τύπους που δηλώνονται.



Σχήμα 2.1 *Αποθήκευση και ανάκληση ASCII χαρακτήρα*

2.2.2. Ο κώδικας (πίνακας) ASCII

Ο κώδικας ASCII αντιστοιχεί χαρακτήρες στις 256 καταστάσεις που μπορεί να περιέχει ένα byte. Η τυπική κωδικοποίηση ASCII χρησιμοποιεί τις πρώτες 128 καταστάσεις, με κωδικούς από το δεκαδικό 0 (δυαδική απεικόνιση σε byte: 00000000) έως το δεκαδικό 127 (δυαδική απεικόνιση σε byte: 01111111). Οι υπόλοιπες 128 καταστάσεις (128 έως 255 ή ισοδύναμα στο δυαδικό σύστημα 10000000 έως 11111111) αποτελούν επέκταση του πρότυπου κώδικα (extended ASCII code) και διαφέρουν ανάλογα με το σύστημα.

Οι πρώτες 31 θέσεις του πίνακα ASCII είναι μη εκτυπούμενοι χαρακτήρες και χρησιμοποιούνται ως χαρακτήρες ελέγχου. Στον **Πίνακα 2.1** απεικονίζεται ο πρότυπος πίνακας ASCII, όπου δίπλα στον κωδικό ASCII εμφανίζεται ο αντίστοιχος χαρακτήρας.

ASCII		ASCII		ASCII		ASCII		ASCII		ASCII		ASCII		ASCII	
0		16		32		48	0	64	@	80	P	96	‘	112	p
1		17		33	!	49	1	65	A	81	Q	97	a	113	q
2		18		34	«	50	2	66	B	82	R	98	b	114	r
3		19		35		51	3	67	C	83	S	99	c	115	s
4		20		36	$	52	4	68	D	84	T	100	d	116	t
5		21		37	%	53	5	69	E	85	U	101	e	117	u
6		22		38	&	54	6	70	F	86	V	102	f	118	v
7		23		39	‘	55	7	71	G	87	W	103	g	119	w
8		24		40	(	56	8	72	H	88	X	104	h	120	x
9		25		41	)	57	9	73	I	89	Y	105	i	121	y
10		26		42	*	58	:	74	J	90	Z	106	j	122	z
11		27		43	+	59	;	75	K	91	[	107	k	123	{
12		28		44	,	60	<	76	L	92	\	108	l	124	|
13		29		45	-	61	=	77	M	93	]	109	m	125	}
14		30		46	.	62	>	78	N	94	^	110	n	126	~
15		31		47	/	63	?	79	O	95	_	111	o	127	

Πίνακας 2.1 *Ο κώδικας ASCII*

2.2.3. Ο τύπος του ακεραίου

Ο τύπος του ακεραίου (`int` από τη λέξη integer) χρησιμοποιείται, για να παραστήσει ακέραιους αριθμούς. Η περιοχή τιμών εξαρτάται από την αρχιτεκτονική του μηχανήματος. Για έναν υπολογιστή που διαθέτει 4 bytes (32 bits) για κάθε ακέραιο, το σύνολο των δυνατών τιμών είναι $2^{32} = 4.284.967.296$. Εάν θεωρηθεί ότι τις

μισές τιμές θα καταλάβουν αρνητικοί ακέραιοι και τις υπόλοιπες μισές θετικοί, η περιοχή τιμών του τύπου **int** είναι από -2.147.483.648 +2.147.483.647, με την ακόλουθη αντιστοιχία δυαδικής και δεκαδικής τιμής:

$(00000000000000000000000000000000)_2 \rightarrow (-2.147.483.648)_{10}$

$(00000000000000000111111111111111)_2 \rightarrow (-1)_{10}$

$(00000000000000001000000000000000)_2 \rightarrow (0)_{10}$

$(11111111111111111111111111111111)_2 \rightarrow (+2.147.483.647)_{10}$

Η δήλωση της μεταβλητής ακεραίου ακολουθεί τον εξής φορμαλισμό:

```
int όνομα_μεταβλητής;   π.χ.   int num;
```

Υπάρχει η δυνατότητα ταυτόχρονα με τη δήλωση να αποδοθεί αρχική τιμή στη μεταβλητή:

```
int num=46;
```

Εάν πριν από τη λέξη **int** τοποθετηθεί ο προσδιοριστής **long**, τότε οι ακέραιοι **long int** εξασφαλίζουν τον ίδιο αποθηκευτικό χώρο με τον απλό ακέραιο. Ωστόσο, στο πρότυπο C99 έχει προστεθεί ο τύπος **long long int**, για τις μεταβλητές του οποίου διατίθεται χώρος 64 bits. Επιπλέον, ο προσδιοριστής **unsigned** χρησιμοποιείται πριν από τη λέξη **int**, για να χαρακτηρίσει τη μεταβλητή χωρίς πρόσημο. Το πεδίο τιμών ενός μη προσημασμένου ακέραιου μήκους *n* bits εκτείνεται από 0 έως $2^n - 1$.

Η εισαγωγή τιμών στις ακέραιες μεταβλητές γίνεται με χρήση της συνάρτησης **scanf()** και του προσδιοριστή **%d** (από τη λέξη *decimal*). Η πρόταση

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

διαβάζει από το πληκτρολόγιο έναν ακέραιο και τον αποδίδει στη μεταβλητή **num**.

Η εκτύπωση ακέραιων μεταβλητών γίνεται με χρήση της συνάρτησης **printf()** και των προσδιοριστών **%d**, **%o**, **%x** για την εμφάνιση σε δεκαδική, οκταδική και δεκαεξαδική μορφή, αντίστοιχα. Οι προσδιοριστές **l** (long), **h** (short), και **u** (unsigned) τοποθετούνται πριν από τους **d**, **o**, **x**. Η πρόταση

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

θα τυπώσει (θεωρώντας ότι η **num** λαμβάνει την τιμή **46**)

```
dec=46, octal=56, hex=2e
```

Παρατηρήσεις:

1. Υπάρχει η δυνατότητα να καθοριστεί ο αριθμός των ψηφίων που θα τυπωθούν, τοποθετώντας τον επιθυμητό αριθμό ανάμεσα στο **%** και το **d**. Εάν ο αριθμός είναι μικρότερος από τον απαιτούμενο αριθμό ψηφίων του ακέραιου, η επιλογή δεν θα ληφθεί υπόψη. Στην αντίθετη περίπτωση, στις πλεονάζουσες θέσεις θα τοποθετηθούν κενά. Για τον λόγο αυτό, ο αριθμός που τοποθετείται στον προσδιοριστή **%d**, ονομάζεται ***καθοριστικό ελάχιστου πλάτους πεδίου***. Με αυτόν τον τρόπο, σε διαδοχικές **printf()** θα υπάρξει ευθυγράμμιση των αποτελεσμάτων κατά στήλες. Οι προτάσεις

```
printf( "dec=%1d,  octal=%4o,  hex=%4x", num,num,num );
printf( "dec=%4d,  octal=%4o,  hex=%4", num,num,num );
```

θα τυπώσουν αντίστοιχα

```
dec=46, octal= 56, hex= 2e
dec= 46, octal= 56, hex= 2e
```

2. Όταν γράφεται ένας αριθμός στον πηγαίο κώδικα χωρίς δεκαδικό ή εκθετικό μέρος, ο μεταγλωττιστής τον χειρίζεται ως ***ακέραια σταθερά***. Η σταθερά **245** αποθηκεύεται ως **int**, ενώ η σταθερά **100000** αποθηκεύεται ως **long int**. Εάν οριστεί η σταθερά **8965** ως **8965LL**, ο μεταγλωττιστής αναγκάζεται να δεσμεύσει χώρο για **long long int**.

2.2.3.1. Παράδειγμα

Να καταστρωθεί πρόγραμμα που να εξετάζει τα μήκη και τα πεδία τιμών των διάφορων παραλλαγών του τύπου ακεραίου.

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

int main()
{
   /* INT_MAX είναι ο μέγιστος int: ορίζεται στο αρχείο κεφαλίδας
   limits.h. Ο τελεστής sizeof δίνει το μέγεθος ενός τύπου δεδομένου
   ή μίας μεταβλητής */
   printf( "The size of type int is %d bytes\n",sizeof(int) );
   printf( "The size of type short is %d bytes\n",sizeof(short) );
   printf( "The size of type long is %d bytes\n",sizeof(long) );
   printf( "The size of type long long is %d bytes\n",sizeof(long
long) );
   printf( "\nmin int: %d,   max int: %d\n",INT_MIN,INT_MAX );
   printf( "\nmin short: %d,   max short: %d\n",SHRT_MIN,SHRT_MAX );
   printf( "\nmin long: %d,   max long: %d\n",LONG_MIN, LONG_MAX );

   return 0;
}
```

```
The size of type int is 4 bytes
The size of type short is 2 bytes
The size of type long is 4 bytes
The size of type long long is 8 bytes

min int: -2147483648,   max int: 2147483648

min short: -32768,   max short: 32767

min long: -2147483648,   max long: 2147483648
```

Εικόνα 2.1 *Η έξοδος του προγράμματος του παραδείγματος 2.2.3.1*

Από τα αποτελέσματα προκύπτει ότι ταυτίζονται τα bytes για `int` και `long`, ενώ ο πλέον εκτεταμένος τύπος ακεραίου `long long int` καταλαμβάνει 8 bytes.

2.2.3.2. Παράδειγμα

Να καταστρωθεί πρόγραμμα που να επιτελεί τα παρακάτω:

Ζήτησε από τον χρήστη έναν χαρακτήρα
Πάρε από τον χρήστη τον χαρακτήρα
Τύπωσε τον χαρακτήρα και τον ASCII κωδικό του
Βρες τον επόμενο χαρακτήρα
Τύπωσέ τον μαζί με τον κωδικό του

```
#include <stdio.h>
int main()
{
```

```
    char charVar,nextChar;
    printf( "\tWrite a character:\t" );
    scanf( "%c",&charVar );
    printf( "\tThe ASCII code of char %c is %d\n", charVar, charVar );
    nextChar = charVar + 1;      /* βρίσκει τον επόμενο χαρακτήρα */
    printf( "\tThe ASCII code of char %c is %d\n", nextChar, nextChar );

    return 0;
}
```

```
Write a character:      D
The ASCII code of character D is 68
The ASCII code of character E is 69
```

Εικόνα 2.2 *Η έξοδος του προγράμματος του παραδείγματος 2.2.3.2*

2.2.4. Τύποι πραγματικών αριθμών

Οι πραγματικοί αριθμοί είναι οι αριθμοί που διαθέτουν κλασματικό μέρος και εκφράζονται συνήθως στις μορφές του **Πίνακα 2.2**:

Αριθμός με δεκαδικά	*Επιστημονική σημειογραφία*	*Εκθετική σημειογραφία*
123.456	1.23456x102	1.23456e+02
0.00002	2.0x10-5	2.0e-5
50000.0	2.0x104	5.0e+04

Πίνακας 2.2 *Σημειογραφίες των πραγματικών αριθμών*

Η γλώσσα C διαθέτει δύο τύπους για αναπαράσταση πραγματικών αριθμών. Τον τύπο **`float`** για αριθμούς κινητής υποδιαστολής απλής ακρίβειας και τον τύπο **`double`** για αριθμούς κινητής υποδιαστολής διπλής ακρίβειας. Ένας νεότερος τύπος εκτεταμένης ακρίβειας είναι ο **`long double`**.

Η δήλωση της μεταβλητής **`float`** ή **`double`** ακολουθεί τον εξής φορμαλισμό:

`float όνομα_μεταβλητής;` π.χ. **`float plank=6.63e-34;`**

Όπως και στις περιπτώσεις των τύπων του χαρακτήρα και του ακεραίου, επιτρέπεται η αρχικοποίηση ταυτόχρονα με τη δήλωση μίας μεταβλητής κινητής υποδιαστολής.

Η εισαγωγή τιμών στις μεταβλητές κινητής υποδιαστολής γίνεται με χρήση της συνάρτησης **`scanf()`** και του προσδιοριστή **`%f`** (από τη φράση *floating point*). Η πρόταση

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

διαβάζει από το πληκτρολόγιο έναν πραγματικό αριθμό και τον αποδίδει στη μεταβλητή **`num`**. Εάν πρόκειται να αναγνωσθεί αριθμός διπλής ακριβείας, απαιτείται η χρήση του προσδιοριστή **`%lf`**, ενώ στην ανάγνωση αριθμού εκτεταμένης ακριβείας ο προσδιοριστής **`%f`** αντικαθίσταται από τον προσδιοριστή **`%Lf`**.

Η εκτύπωση πραγματικών μεταβλητών γίνεται με χρήση της συνάρτησης **`printf()`** και των προσδιοριστών **`%f`** για εμφάνιση σε δεκαδική μορφή, **`%e`** για εμφάνιση σε εκθετική μορφή και **`%g,`** για να ανατεθεί στο σύστημα να επιλέξει μεταξύ των δύο προηγουμένων, με προτεραιότητα στη μορφή με το μικρότερο μέγεθος.

Σε ό,τι αφορά τον χώρο που καταλαμβάνουν στη μνήμη, ως συνηθισμένα μεγέθη αναφέρονται για τους μεν **`float`** τα 32 bits, για τους δε **`double`** τα 64 bits. Θα πρέπει να σημειωθεί ότι, σε αντιδιαστολή με τους ακέραιους αριθμούς, δεν υπάρχει αντιστοιχία ένα προς ένα ανάμεσα στους πραγματικούς αριθμούς και στις απεικονίσεις τους στις γλώσσες προγραμματισμού. Οι αριθμοί που αντιστοιχούν στους πραγματικούς αριθμούς είναι προσεγγίσεις αυτών και ονομάζονται ***αριθμοί μηχανής***, εξαιτίας της ανάγκης να

απεικονιστούν με πεπερασμένο αριθμό ψηφίων πραγματικοί αριθμοί που θεωρητικά μπορούν να περιέχουν άπειρο αριθμό κλασματικών ψηφίων. Σε μία μεταβλητή τύπου **`float`** των 32 bits, τα 8 bits χρησιμοποιούνται για τον εκθέτη, ένα για το πρόσημο και τα υπόλοιπα 23 για το κλασματικό μέρος, που ονομάζεται **mantissa**. Η μορφή του αριθμού είναι η ακόλουθη:

$$\pm(\ .\ d_1 d_2 ... d_{23}\ ) \cdot 2^e$$

όπου τα ψηφία $d_1 ... d_{23}$ είναι δυαδικά και το e είναι το δεκαδικό ισοδύναμο του οκταψήφιου δυαδικού εκθέτη. Κατά σύμβαση, $d_1 = 1$. Το δεκαδικό ισοδύναμο της ανωτέρω μορφής είναι:

$$\pm\left[(d_1 \cdot 2^{-1}) + (d_2 \cdot 2^{-2}) + ... + (d_{23} \cdot 2^{-23})\right] \cdot 2^e = \pm 2^e \cdot \sum_{i=1}^{23} d_i \cdot 2^{-i}$$

Κατά συνέπεια, με βάση το παραπάνω σύστημα απεικόνισης, το μέγεθος της κλασματικής ακριβείας ενός αριθμού κινητής υποδιαστολής καθορίζεται από τον αριθμό των κλασματικών ψηφίων.

Ο οκταψήφιος δυαδικός εκθέτης αντιστοιχεί σε $2^8 = 256$ δυνατές τιμές. Από αυτές οι 127 δίνονται σε αρνητικούς ακεραίους και οι υπόλοιπες 129 σε θετικούς, με την ακόλουθη αντιστοιχία δυαδικής και δεκαδικής τιμής:

$$e_{\min} = (00000000)_2 \quad \rightarrow \quad e_{\min} = (-127)_{10}$$
$$e_{\max} = (11111111)_2 \quad \rightarrow \quad e_{\max} = (128)_{10}$$

Με βάση τα παραπάνω, η μέγιστη απόλυτη τιμή πραγματικών αριθμών που μπορεί να επιτευχθεί με μεταβλητή τύπου **`float`** των 32 bits είναι:

$$|\max| = \left[(1 \cdot 2^{-1}) + (1 \cdot 2^{-2}) + ... + (1 \cdot 2^{-23})\right] \cdot 2^{128} = 3.402823e + 38$$

Στους αριθμούς διπλής ακριβείας δεσμεύονται 64 bits, εκ των οποίων τα 11 δίνονται στον εκθέτη, ένα στο πρόσημο και 52 στο κλασματικό μέρος.

<u>Παρατηρήσεις:</u>

1. Όπως σημειώθηκε στους ακέραιους, έτσι και στους πραγματικούς υπάρχει η δυνατότητα να καθορισθεί ο αριθμός των ψηφίων που θα εκτυπωθούν, τοποθετώντας τον επιθυμητό αριθμό ανάμεσα στο **`%`** και το **`f`**. Μάλιστα ο αριθμός θα είναι της μορφής **`a.b`**, με το **`a`** να δηλώνει τον συνολικό αριθμό των ψηφίων– συμπεριλαμβανομένου του προσήμου– και το **`b`** να δηλώνει τον αριθμό των δεκαδικών ψηφίων. Οι προτάσεις

```
float num=46.37;
printf( "num=%8.4f,  num=%12.1f\n", num,num );
```

οδηγούν στην εμφάνιση στην οθόνη των ακόλουθων αποτελεσμάτων:

```
num= 46.3700, num=       46.4
```

Είναι φανερό ότι στην περίπτωση που ο αριθμός των δεκαδικών ψηφίων που ζητούνται είναι μικρότερος από τον απαιτούμενο, γίνεται στρογγυλοποίηση (το **37** στρογγυλοποιείται στο **40** και παρελείπεται το **0**).

2. Πραγματικοί αριθμοί, όπως οι:

```
0.12      45.68      9e-5      24e09      0.0034e-08
```

όταν εμφανίζονται στον πηγαίο κώδικα αποτελούν τις ***πραγματικές σταθερές***. Θεωρούνται από τον μεταγλωττιστή ως **`double`** και δεσμεύουν τον αντίστοιχο χώρο.

2.2.4.1. Παράδειγμα

Να καταστρωθεί πρόγραμμα που να εξετάζει τα μήκη και τα πεδία τιμών των τύπων κινητής υποδιαστολής μονής και διπλής ακρίβειας.

```
#include <stdio.h>
#include <float.h>       /* για τα όρια του float */
int main()
{
```

```
    printf( "The size of type float is %d bytes\n",sizeof(float) );
    printf( " The size of type double is %d bytes\n",sizeof(double) );
    printf( "\nmin float: %e,   max float: %e\n",FLT_MIN,FLT_MAX);
    printf( "\nmin double: %e,   max double: %e\n",DBL_MIN,DBL_MAX);
    printf( "\n\nPress any key to continue" );

    return 0;
}
```

```
The size of type float is 4 bytes
The size of type double is 8 bytes

min float: 1.175494e-038,   max float: 3.402823e+038

min double: 2.225047e-308,   max double: 1.797693e+308
```

Εικόνα 2.3 *Η έξοδος του προγράμματος του παραδείγματος 2.2.4.1*

2.2.5 Ο προσδιοριστής const

Στη γλώσσα C μπορούν να οριστούν και να αρχικοποιηθούν μεταβλητές, για τις οποίες δεν επιτρέπεται η μεταβολή της τιμής τους. Τέτοιες μεταβλητές δηλώνονται, όπως και οι υπόλοιπες, με την προσθήκη του προσδιοριστή **`const`** στα αριστερά του τύπου τους:

```
const double numDouble=46.358767;
const int numInt=46;
const int numChar='G';
```

Στις παραπάνω προτάσεις δηλώνονται και αρχικοποιούνται οι μεταβλητές τριών διαφορετικών τύπων **`numDouble`**, **`numInt`** και **`numChar`**. Οι μεταβλητές αυτές δεν μπορούν να λάβουν άλλη τιμή καθ'όλη τη διάρκεια εκτέλεσης του προγράμματος.

2.3. Είσοδος/Έξοδος προγράμματος

Η είσοδος/έξοδος προγράμματος (input/output) αναφέρεται στις λειτουργίες που γίνονται στις συσκευές εισόδου (συνήθως το πληκτρολόγιο) και εξόδου (συνήθως η οθόνη) του υπολογιστή. Η γλώσσα C δεν διαθέτει εντολή επικοινωνίας με το εξωτερικό περιβάλλον και το έργο αυτό το επιτελούν συναρτήσεις. Προηγουμένως, χρησιμοποιήθηκε το ζεύγος των **`scanf()`** και **`printf()`**, οι οποίες αποτελούν τις μορφοποιούμενες I/O κονσόλας, διότι μπορούν να διαβάζουν και να τυπώνουν δεδομένα σε διάφορες μορφές. Πέραν αυτών, οι συναρτήσεις **`getch()`**, **`getchar()`**, **`putchar()`** διαβάζουν και τυπώνουν με απλούστερο τρόπο δεδομένα τύπου χαρακτήρα, χωρίς όμως να διαθέτουν δυνατότητες μορφοποίησης.

2.3.1. Η συνάρτηση printf()

Στη γλώσσα C η έξοδος των δεδομένων προς το εξωτερικό περιβάλον (συσκευές εξόδου) γίνεται μέσω ροών (αρχείων) εξόδου. Ως *προκαθορισμένη ή πρότυπη ροή εξόδου* (**st**andar**d** **out**put stream – stdout) έχει τεθεί η οθόνη. Η συνάρτηση **`printf()`** (**print f**ormatted data) χρησιμοποιείται για την εμφάνιση μορφοποιούμενης πληροφορίας στην οθόνη. Τυπικά η **`printf()`** εμφανίζει στην οθόνη το περιεχόμενο του ορίσματός της, που περικλείεται μέσα στις παρενθέσεις που την ακολουθούν. Το πρωτότυπο της συνάρτησης έχει την ακόλουθη μορφή:

```
int printf(const char *content, argument_1, argument_2, ..., argument_n);
```

Η συνάρτηση **printf()** εκτυπώνει κατά βάση μία συμβολοσειρά, η οποία εναλλακτικά καλείται αλφαριθμητικό (στο ανωτέρω πρωτότυπο της **printf()** το αλφαριθμητικό το διαχειρίζεται ο δείκτης **content** – περισσότερες πληροφορίες για τους δείκτες στο Κεφάλαιο 6). Ωστόσο, η ισχύς της **printf()** – όπως διαπιστώθηκε στις προηγούμενες παραγράφους– έγκειται στη δυνατότητα που προσφέρει να αντικατασταθούν τμήματα του αλφαριθμητικού από δεδομένα μέσω των *προσδιοριστών* ή *προσδιοριστικών μετατροπής* (conversion specifiers) **%d**, **%c**, **%f** κ.λπ. Οι τιμές των δεδομένων, τα οποία αντιστοιχούν στα ορίσματα **argument_1**, ..., **argument_n**, μπορούν να απεικονιστούν με συγκεκριμένο αριθμό ψηφίων. Επιπρόσθετα, παρέχεται η δυνατότητα να εκτελεστούν περαιτέρω λειτουργίες μορφοποίησης, όπως η αλλαγή γραμμής ή η μετακίνηση κατά μία θέση στηλοθέτη.

Σε περίπτωση επιτυχημένης εκτέλεσης, η συνάρτηση **printf()** επιστρέφει τον αριθμό των χαρακτήρων που εκτυπώνει, αλλιώς επιστρέφει αρνητική τιμή. Στην παρακάτω γραμμή κώδικα

```
x=printf( "This is test no %6d",213 );
```

η ακέραια μεταβλητή **x** δέχεται τον αριθμό των χαρακτήρων που εκτύπωσε η **printf()**, δηλαδή τον αριθμό **22**, καθόσον το αλφαριθμητικό **"This is test no "** περιέχει 17 χαρακτήρες και ο προσδιοριστής **%6d** απαίτησε να εκτυπωθεί ο ακέραιος **213** με 6 χαρακτήρες (3 κενά και ακολούθως οι χαρακτήρες **2**, **1** και **3**).

Για να χρησιμοποιηθεί η συνάρτηση **printf()** απαιτείται η ενσωμάτωση στο πρόγραμμα– μέσω της εντολής προεπεξεργαστή **include**– του αρχείου κεφαλίδας **stdio.h**, καθώς σε αυτό το αρχείο βρίσκεται το πρότυπό της.

2.3.1.1. Μη εκτυπούμενοι χαρακτήρες και σημαίες

Οι σταθερές τύπου χαρακτήρα ***«νέα γραμμή* (new-line)»** και ***«στηλοθέτης* (tab)»** ανήκουν στην κατηγορία των μη εκτυπούμενων χαρακτήρων, τους οποίους η C αναπαριστά με τις ***«ακολουθίες διαφυγής* (escape sequences)»** **\n** και **\t** αντίστοιχα. Η παρακάτω πρόταση δίνεται ως παράδειγμα χρήσης χαρακτήρων διαφυγής:

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

Η πρόταση θα εμφανίσει στην κύρια έξοδο (οθόνη):

```
Write, " a \ is a backslash. "
```

Οι μη εκτυπούμενοι χαρακτήρες και οι αντίστοιχες ακολουθίες διαφυγής παρατίθενται στον **Πίνακα 2.3**:

Χαρακτήρας	Ακολουθία	Χαρακτήρας	Ακολουθία
συναγερμός (κουδούνι)	**\a**	πλάγια γραμμή	****
οπισθοχώρηση μία θέση	**\b**	λατινικό ερωτηματικό	**\?**
αλλαγή σελίδας	**\f**	μονό εισαγωγικό	**\'**
νέα γραμμή	**\n**	διπλό εισαγωγικό	**\"**
επαναφορά κεφαλής	**\r**	οκταδικός αριθμός	**\ooo**
οριζόντιος στηλοθέτης	**\t**	δεκαεξαδικός αριθμός	**\xhhh**
κατακόρυφος στηλοθέτης	**\v**	μηδενικός χαρακτήρας(με ASCII κωδικό 0)	**\0**

Πίνακας 2.3 *Μη εκτυπούμενοι χαρακτήρες και αντίστοιχες ακολουθίες διαφυγής*

Ένα επιπλέον εργαλείο μορφοποίησης των εκτυπούμενων δεδομένων αποτελούν οι *σημαίες* (flags), που παρατίθενται στον **Πίνακα 2.4**. Αριστερά από κάθε σημαία απαιτείται η προσθήκη του συμβόλου **%**.

Σημαία	Λειτουργία
-	Αριστερή στοίχιση της εξόδου στο πεδίο πλάτους (η προκαθορισμένη στοίχιση είναι στα δεξιά)
+	Προσθήκη του προσήμου στις θετικές τιμές

#o	Προσθήκη του **0** μπροστά από οκταδικούς
#x	Προσθήκη του **0x** μπροστά από δεκαεξαδικούς αριθμούς
#X	Προσθήκη του **0X** μπροστά από δεκαεξαδικούς αριθμούς
0	Προσθήκη των απαιτούμενων μηδενικών μπροστά από την τιμή, ώστε να καλυφθεί το πεδίο πλάτους.
κενός χαρακτήρας	Προσθήκη του κενού χαρακτήρα μπροστά από τις μηδενικές τιμές

Πίνακας 2.4 *Σημαίες*

2.3.1.2. Παράδειγμα

Στο πρόγραμμα που ακολουθεί χρησιμοποιούνται διάφορες σημαίες:

```
#include <stdio.h>

int main()
{
   int x=323;

   printf( "1:\t%-5d\n",x );
   printf( "2:\t%+5d\n",x );
   printf( "3:\t% d\n",x );
   printf( "4:\t%#o\n",x );
   printf( "5:\t%#x\n",x );
   printf( "6:\t%#X\n",x );
   printf( "7:\t%05d\n",x );

   return 0;
}
```

```
1:      323
2:      +323
3:      323
4:      0503
5:      0x143
6:      0X143
7:      00323
```

Εικόνα 2.4 *Η έξοδος του προγράμματος του παραδείγματος 2.3.1.2*

Κάθε `printf()` αρχικά εμφανίζει στην οθόνη τους ακέραιους αριθμούς **1**,**2**,...,**7**, ακολουθούμενους από τον χαρακτήρα `':'`. Στη συνέχεια, εμφανίζεται το περιεχόμενο της ακέραιας μεταβλητής `x`, όπως αυτό μορφορποιείται σύμφωνα με την εκάστοτε σημαία. Συγκεκριμένα:

- Η πρώτη `printf()` εμφανίζει την τιμή **323** στοιχισμένη στα αριστερά.
- Στη δεύτερη `printf()` το πλάτος πεδίου είναι 5 και γίνεται χρήση της σημαίας `+`. Εφόσον το **323** καταλαμβάνει 3 θέσεις, στην οθόνη εμφανίζεται ένας κενός χαρακτήρας και ακολούθως το **+323**.
- Στην τρίτη `printf()` η χρήση της σημαίας κενού χαρακτήρα οδηγεί στην εμφάνιση ενός κενού αριστερά της τιμής **323**.
- Στην τέταρτη `printf()` το **323** εμφανίζεται σε οκταδική μορφή.
- Στην πέμπτη και έκτη `printf()` το **323** εμφανίζεται σε δεκαεξαδική μορφή.
- Στην τελευταία `printf()` προστίθενται δύο μηδενικά πριν το **323**.

2.3.2. Η συνάρτηση scanf()

Αντίστοιχα με την έξοδο δεδομένων, και η είσοδος πληροφορίας από το εξωτερικό περιβάλον (συσκευές εισόδου) γίνεται μέσω ροών εισόδου. Ως *προκαθορισμένη* ή *πρότυπη ροή εισόδου* (**st**andar**d** **in**put stream – stdin) έχει τεθεί το πληκτρολόγιο. Η συνάρτηση **`scanf()`** (**scan** **f**ormatted data) χρησιμοποιείται για την ανάγνωση δεδομένων από το πληκτρολόγιο και την αποθήκευσή τους σε μεταβλητές, σύμφωνα με τους προσδιοριστές που αντιστοιχούν στα ορίσματα της συνάρτησης. Το πρωτότυπο της συνάρτησης έχει την ακόλουθη μορφή:

```
int scanf(const char *content, argument_1, argument_2, ..., argument_n);
```

Θα πρέπει να προσεχθεί ότι τα ορίσματα **`argument_1`**, ..., **`argument_n`**, δεν είναι μεταβλητές αλλά οι διευθύνσεις μνήμης των μεταβλητών, στις οποίες θα αποθηκευτούν τα δεδομένα. Το αλφαριθμητικό μορφοποίησης που διαχειρίζεται ο **`content`** περιέχει τόσα προσδιοριστικά όσα είναι και τα προς ανάγνωση δεδομένα. Αντίστοιχος είναι και ο αριθμός των ορισμάτων **`argument_1`**, ..., **`argument_n`**. Κατά συνέπεια, η ακόλουθη γραμμή κώδικα

```
scanf( "%d %d",&x,&y );
```

θα έχει ως αποτέλεσμα να αναγνωσθούν δύο ακέραιοι και να αποθηκευτούν στις θέσεις μνήμες που καταλαμβάνουν οι ακέραιες μεταβλητές **`x`** και **`y`**. Για να γίνει η εκχώρηση τιμής στη μεταβλητή, επιβάλλεται η εισαγωγή του τελεστή διεύθυνσης **`&`** πριν το όνομα της μεταβλητής (με εξαίρεση τα ονόματα αλφαριθμητικών, στα οποία θα αναφερθούμε στο Κεφάλαιο 5).

Σε περίπτωση επιτυχημένης εκτέλεσης, η συνάρτηση **`scanf()`** επιστρέφει τον αριθμό των αναγνωσθέντων δεδομένων. Εάν κάποιο δεδομένο δεν μπορεί να αναγνωσθεί, η **`scanf()`** τερματίζεται και οι μη αναγνωσθείσες τιμές παραμένουν στο **stdin**. Το πρωτότυπο της συνάρτησης βρίσκεται στο αρχείο κεφαλίδας **`stdio.h`**.

Η συνάρτηση **`scanf()`** απαιτεί προσοχή στη χρήση της, καθώς παρουσιάζει ορισμένα ιδιαίτερα χαρακτηριστικά:

- Η κλήση της συνάρτησης **`scanf()`** διακόπτει τη ροή εκτέλεσης του προγράμματος, έως ότου ο χρήστης πληκτρολογήσει τα δεδομένα και πατήσει το πλήκτρο επαναφοράς (**ENTER**).
- Εάν στο τέλος του αλφαριθμητικού μορφοποίησης προστεθεί η ακολουθία διαφυγής αλλαγής γραμμής (**`'\n'`**), τότε η συνάρτηση **`scanf()`** υποχρεώνεται να προχωρήσει στην ανάγνωση του επόμενου μη κενού χαρακτήρα. Κατά συνέπεια, η ακόλουθη γραμμή κώδικα

```
scanf( "%d\n",&x );
```

οδηγεί στην ανάγνωση ενός ακεραίου και στην παύση της εκτέλεσης του προγράμματος, καθώς αναμένεται η εισαγωγή από τον χρήστη ενός μη κενού χαρακτήρα.

- Όταν η συνάρτηση **`scanf()`** χρησιμοποιείται για την ανάγνωση αριθμητικών τιμών, οι «λευκοί χαρακτήρες» που πιθανόν υπάρχουν πριν την αριθμητική τιμή (οι χαρακτήρας του κενού, της αλλαγής γραμμής και του στηλοθέτη), αγνοούνται.
- Επιτρέπεται η χρήση προσδιοριστή πλάτους μέσα στο αλφαριθμητικό μορφοποίησης. Κατά συνέπεια, στην ακόλουθη γραμμή κώδικα

```
scanf( "%4d",&x );
```

εάν ο χρήστης πληκτρολογήσει **57937425**, θα αποθηκευτούν στην ακέραια μεταβλητή **`x`** οι τιμές των πρώτων τεσσάρων ψηφίων, δηλαδή το περιεχόμενο της **`x`** θα γίνει **5793**.

- Η συνάρτηση **`scanf()`** μπορεί να λάβει προδιαγραφές μετατροπής, δηλαδή να θέσει περιορισμούς στους χαρακτήρες που μπορεί να αναγνώσει, μέσω του προσδιοριστή **`[]`**. Επιπλέον, με χρήση του συμβόλου **`-`** μπορούν να προσδιοριστούν ομάδες αποδεκτών χαρακτήρων, ενώ με χρήση του συμβόλου **`^`** μπορούν να αποκλειστούν χαρακτήρες. Για παράδειγμα, ο προσδιοριστής μορφοποίησης **`%[K-W]`** επιτρέπει την

καταχώρηση μόνο των χαρακτήρων **'K'**,**'L'**,...,**'W'**, ενώ ο προσδιοριστής **%[^K-W]** δεν επιτρέπει την ανάγνωση των χαρακτήρων αυτών.

- Εάν μπροστά από έναν προσδιοριστή τοποθετηθεί αστερίσκος *****, τότε το δεδομένο που θα δοθεί από το πληκτρολόγιο θα αναγνωσθεί μεν, ωστόσο δεν θα αποθηκευτεί στο αντίστοιχο όρισμα της συνάρτησης.

- Η συνάρτηση **scanf()** μπορεί να εμφανίσει προβληματική συμπεριφορά, όταν επιχειρηθεί η ανάγνωση δεδομένων τύπου χαρακτήρα με διαδοχικές κλήσεις της. Αυτό οφείλεται στο ότι κατά την πληκτρολόγηση του **ENTER**, ο χαρακτήρας αλλαγής γραμμής αποθηκεύεται στο **stdin** και η επόμενη κλήση της **scanf()** τον ανασύρει και τον αποθηκεύει στο όρισμα εκείνο, το οποίο ανέμενε να αποθηκεύσει τον χαρακτήρα που θα πληκτρολογούσε ο χρήστης.

2.3.2.1. Παράδειγμα

Στο πρόγραμμα που ακολουθεί αποτυπώνεται η χρήση της συνάρτησης **scanf()** με διάφορους τρόπους:

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

int main()
{
   int v1,v2;
   char c1,c2;
   /* 1η ομάδα προτάσεων */
   printf( "\n\tGive two integers: " );
   scanf(" \t%d,%d",&v1,&v2 );
   printf(" \n\tv1=%d, v2=%d",v1,v2 );
   /* 2η ομάδα προτάσεων */
   printf( "\n\n\tGive a character: " );
   scanf( "\t%c",&c1 );
   printf( "\n\tc1=%c",c1 );
   /* 3η ομάδα προτάσεων */
   printf( "\n\n\tGive another integer: " );
   scanf( "\t%d",&v1 );
   printf( "\n\tv1=%d",v1 );
   /* 4η ομάδα προτάσεων */
   printf( "\n\n\tGive two new characters: " );
   scanf( "%c",&c1 );
   scanf( "%c",&c2 );
   printf( "\n\tc1=%c\tc2=%c",c1,c2 );

   return 0;
}
```

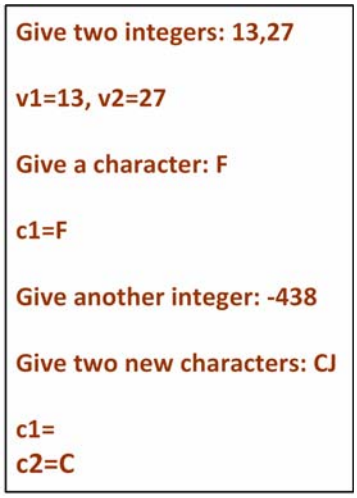
- Στην πρώτη ομάδα προτάσεων η συνάρτηση **scanf()** διαβάζει δύο ακεραίους, τις τιμές των οποίων αποδίδει στις ακέραιες μεταβλητές **v1** και **v2**.

- Στη δεύτερη ομάδα προτάσεων αναγιγνώσκεται ένας χαρακτήρας και αποθηκεύεται στη μεταβλητή τύπου χαρακτήρα **c1**.

- Στην τρίτη ομάδα προτάσεων γίνεται η ανάγνωση και αποθήκευση ενός ακεραίου.

- Στην τέταρτη ομάδα προτάσεων αναδεικνύεται το πρόβλημα στη λειτουργία της **scanf()** που διατυπώθηκε προηγουμένως: ενώ επιχειρείται να αναγνωσθούν οι χαρακτήρες **'C'**,...,**'J'**, στη μεταβλητή **c1** αποθηκεύεται ένας κενός χαρακτήρας και στη μεταβλητή **c2** αποθηκεύεται το **'C'**. Το παράδοξο αποτέλεσμα εξηγείται από το γεγονός ότι στο τέλος της τρίτης σειράς προτάσεων, η πληκτρολόγηση του **ENTER** μετά την τιμή **-438** οδήγησε την αποθήκευση του χαρακτήρα αλλαγής γραμμής στο stdin. Όταν,

ακολούθως, πληκτρολογήθηκαν οι χαρακτήρες `'C'`,...,`'J'`, αυτοί τοποθετήθηκαν με τη σειρά στο **stdin**. Έτσι, όταν έγιναν οι εκχωρήσεις τιμής, πρώτα αποδόθηκε στη μεταβλητή `c1` ο αποθηκευθείς λευκός χαρακτήρας της αλλαγής γραμμής, μετά αποθηκεύτηκε στη μεταβλητή `c2` ο χαρακτήρας `'C'`, ενώ ο χαρακτήρας `'J'` παρέμεινε στο **stdin**.



Εικόνα 2.5 *Η έξοδος του προγράμματος του παραδείγματος 2.3.2.1*

2.3.3. Η συνάρτηση getch()

Η συνάρτηση `getch()` διαβάζει έναν χαρακτήρα από την προκαθορισμένη είσοδο. Αναμένει, έως ότου πατηθεί ένα πλήκτρο, και στη συνέχεια επιστρέφει την τιμή του, χωρίς όμως να εμφανίζεται στην οθόνη το πλήκτρο που πατήθηκε. Το πρωτότυπο της συνάρτησης `getch()` βρίσκεται στο αρχείο κεφαλίδας `stdio.h` και η δήλωσή της είναι

```
int getche(void);
```

Η συνάρτηση `getch()` επιστρέφει μεν έναν ακέραιο, αλλά μόνο στο πρώτο byte (byte χαμηλής τάξης) περιέχεται ο χαρακτήρας. Η χρήση ακεραίων γίνεται για λόγους συμβατότητας με τον αρχικό μεταγλωττιστή της UNIX C.

2.3.4. Οι συναρτήσεις getchar() – putchar()

Οι συναρτήσεις `getchar()` (**get char**acter) και `putchar()` (**put char**acter) είναι το αρχικό ζεύγος εισόδου– εξόδου χαρακτήρων και βασίζονται στο UNIX. Τα πρωτότυπά τους βρίσκονται στο αρχείο κεφαλίδας `stdio.h` και οι δηλώσεις τους είναι

```
int getchar(void);
int putchar(int c);
```

Η συνάρτηση `getchar()` διαβάζει έναν χαρακτήρα από την προκαθορισμένη είσοδο και τον επιστρέφει στο πρόγραμμα. Αποτελεί παραλλαγή της `getch()`. Ένα μειονέκτημα της συνάρτησης αυτής είναι ότι κρατά την είσοδο στην περιοχή προσωρινής αποθήκευσης, μέχρι να πατηθεί **ENTER**. Έτσι, μετά την επιστροφή της `getchar()` περιμένουν ένας ή περισσότεροι χαρακτήρες στην ουρά εισόδου του **stdin**.

Η συνάρτηση `putchar()` εμφανίζει στην οθόνη τον χαρακτήρα που έχει ως όρισμα, στην τρέχουσα θέση του δρομέα. Επιστρέφει μεν έναν ακέραιο, αλλά το byte χαμηλής τάξης είναι αυτό που περιέχει τον χαρακτήρα. Όπως συνέβη και με τις προηγούμενες συναρτήσεις, η χρήση ακεραίων γίνεται για λόγους συμβατότητας με τον αρχικό μεταγλωττιστή της UNIX C.

2.3.4.1. Παράδειγμα

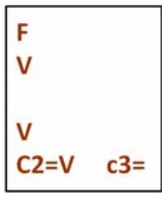
Στο πρόγραμμα που ακολουθεί αποτυπώνεται η χρήση των συναρτήσεων **getch()**, **getchar()** και **putchar()**:

```
#include <stdio.h>

int main()
{
   char c1,c2,c3;

   c1=getch();
   putchar(c1);
   printf( "\n" );
   c2=getchar();
   printf( "\n" );
   putchar(c2);
   c3=getchar();
   printf( "\nc2=%c\tc3=%c",c2,c3 );

   return 0;
}
```



Εικόνα 2.6 *Η έξοδος του προγράμματος του παραδείγματος 2.3.4.1*

Η συνάρτηση **getch()** διαβάζει τον χαρακτήρα **'F'** και τον αποδίδει στη μεταβλητή **c1**, αλλά δεν τον εμφανίζει. Η εμφάνισή του οφείλεται στη χρήση της **putchar()** με όρισμα τη μεταβλητή **c1**. Ακολούθως, η **getchar()** διαβάζει τον χαρακτήρα **'V'** και τον εμφανίζει στην οθόνη. Η δεύτερη εμφάνιση του **'V'** είναι αποτέλεσμα της **putchar(c2)**.

Η πρόταση **c3=getchar()** θα έπρεπε να έχει ως αποτέλεσμα την εκ νέου ανάγνωση χαρακτήρα, ωστόσο τα αποτελέσματα δεν είναι τα αναμενόμενα: από την προηγούμενη χρήση της **getchar()** παρέμενε αποθηκευμένος στον χώρο προσωρινής αποθήκευσης ο λευκός χαρακτήρας αλλαγής γραμμής, που αντιστοιχεί στο πλήκτρο **ENTER**. Αυτός ο χαρακτήρας αποδόθηκε στη μεταβλητή **c3**, γεγονός που αναδεικνύει το ζήτημα που ανακύπτει σε διαδοχικές εκχωρήσεις τιμών σε μεταβλητές με χρήση της **getchar()**.

Ένας τρόπος να ξεπεραστεί το πρόβλημα, είναι να παρεμβληθεί η εντολή **fflush(stdin)** ανάμεσα στις προτάσεις **c2=getchar()** και **c3=getchar()**. Η **fflush()** καθαρίζει τον χώρο προσωρινής αποθήκευσης. Ωστόσο, επειδή σύμφωνα με το πρότυπο της γλώσσας C η συμπεριφορά της **fflush()** είναι απροσδιόριστη, μία άλλη λύση είναι να παρεμβληθεί μία **getchar()** χωρίς να εκχωρεί το αναγνωσθέν δεδομένο σε κάποια μεταβλητή, ώστε να διαβάσει τον αποθηκευμένο χαρακτήρα αλλαγής γραμμής:

```
#include <stdio.h>

int main()
{
```

```
    char c1,c2,c3;

    c1=getch();
    putchar(c1);
    printf( "\n" );
    c2=getchar();
    printf( "\n" );
    putchar(c2);
    getchar();
    c3=getchar();
    printf( "\nc2=%c\tc3=%c",c2,c3 );

    return 0;
}
```

```
F
V

VG
C2=V    c3=G
```

Εικόνα 2.7 *Η νέα έξοδος του προγράμματος του παραδείγματος 2.3.4.1*

2.3.5. Η συνάρτηση kbhit()

Η συνάρτηση **kbhit()** (**k**ey**b**oard **hit**) ελέγχει κατά πόσο ο χρήστης έχει πατήσει κάποιο πλήκτρο. Εφόσον έχει πατήσει κάποιο πλήκτρο, η συνάρτηση επιστρέφει ως αληθής (δηλαδή επιστρέφεται μη μηδενικός ακέραιος), σε αντίθετη περίπτωση επιστρέφει ως ψευδής (δηλαδή επιστρέφεται το μηδέν). Η συνάρτηση **kbhit()** χρησιμοποιείται κυρίως για να διακόπτει ο χρήστης τη ροή του προγράμματος κατά το δοκούν.

Το πρωτότυπο της **kbhit()** βρίσκεται στο αρχείο κεφαλίδας **stdio.h** και η δήλωσή της είναι

```
int kbhit(void);
```

2.3.6. Η συνάρτηση exit()

Η συνάρτηση **exit()** επιτρέπει τον άμεσο τερματισμό ενός προγράμματος. Οι προτάσεις που βρίσκονται κάτω από την **exit()** δε θα εκτελεστούν. Το πρωτότυπο της **exit()** βρίσκεται στο αρχείο κεφαλίδας **stdlib.h** και η δήλωσή της είναι

```
void exit(int status);
```

Η μεταβλητή **status** υποδηλώνει κατά πόσο ο τερματισμός είναι κανονικός ή όχι. Σε περίπτωση κανονικού τερματισμού η **exit()** καλείται ως

```
exit(0);        ή      exit(EXIT_SUCCESS);
```

ενώ στην περίπτωση τερματισμού λόγω σφάλματος καλείται ως

```
exit(EXIT_FAILURE);
```

Θα πρέπει να σημειωθεί ότι το πρότυπο της γλώσσας C συσχετίζει μόνο το **EXIT_FAILURE** με την περίπτωση τερματισμού λόγω σφάλματος, ωστόσο από τους προγραμματιστές χρησιμοποιούνται και μη μηδενικοί ακέραιοι (συνήθως **1** και **-1**) για να υποδηλώσουν τις ενδεχόμενες περιπτώσεις σφάλματος.

2.4. Η έννοια της έκφρασης και του τελεστή στη γλώσσα C

Στον προγραμματισμό οι τελεστές (operators) έχουν την ίδια έννοια που έχουν και στα μαθηματικά, αποτελώντας σύμβολα ή λέξεις που αναπαριστούν συγκεκριμένες διεργασίες, οι οποίες εκτελούνται επί ενός ή περισσότερων δεδομένων. Τα δεδομένα καλούνται **τελεστέοι** (operands) και μπορούν να είναι μεταβλητές, σταθερές ή ακόμη και κλήσεις συναρτήσεων.

Οι τελεστές χρησιμοποιούνται για τον σχηματισμό εκφράσεων. *Μία έκφραση, εν γένει, αποτελείται από έναν ή περισσότερους τελεστέους και από έναν ή περισσότερους τελεστές. Κάθε έκφραση έχει μία τιμή, η οποία υπολογίζεται με ορισμένους κανόνες*. Για παράδειγμα, στην έκφραση `num+12` ο χαρακτήρας `+` αναπαριστά τη διεργασία της πρόσθεσης των δύο τελεστέων, οι οποίοι είναι η μεταβλητή `num` και η σταθερά **12**.

Οι τελεστές ταξινομούνται, ανάλογα με τον αριθμό των τελεστέων στους οποίους δρουν, σε **μοναδιαίους** (unary), **δυαδικούς** (binary) και **τριαδικούς** (ternary). Μία δεύτερη κατηγοριοποίηση επιτελείται με βάση τη διεργασία που εκτελούν, οδηγώντας στις κατηγορίες του **Πίνακα 2.5**:

Κατηγορία	Ενδεικτικοί τελεστές
αριθμητικοί	`+ - * /`
λογικοί	`&& || !`
συσχετιστικοί	`> >= = = !=`
διαχείρισης bits	`>> & ! ^`
διαχείρισης μνήμης	`& [ ] . ->`

Πίνακας 2.5 *Κατηγορίες τελεστών*

Τα σύμβολα των συνηθέστερων δυαδικών τελεστών στη C παρατίθενται στον **Πίνακα 2.6**:

Δυαδικός τελεστής	Σύμβολο	Δυαδικός τελεστής	Σύμβολο
μικρότερο	`<`	πρόσθεση	`+`
μικρότερο ή ίσο	`<=`	αφαίρεση	`-`
ίσο	`= =`	πολλαπλασιασμός	`*`
διάφορο	`!=`	διαίρεση πραγματικών	`/`
μεγαλύτερο	`>`	πηλίκο διαίρεσης ακεραίων	`/`
μεγαλύτερο ή ίσο	`>=`	υπόλοιπο διαίρεσης ακεραίων	`%`

Πίνακας 2.6 *Σύμβολα δυαδικών τελεστών*

2.4.1. Κατηγορίες τελεστών– σημειογραφία

Η γλώσσα C δίνει τη δυνατότητα να επιτυγχάνονται διαφορετικές λειτουργίες στην ίδια έκφραση ανάλογα με τη θέση των τελεστών ανάμεσα στους τελεστέους. Για τον λόγο αυτό, έχουν αναπτυχθεί τρεις σημειογραφίες για τους δυαδικούς τελεστές:

- Η σημειογραφία ***ένθεσης*** ή ***ένθετου τελεστή*** (infix notation), όταν ο τελεστής τοποθετείται μεταξύ των τελεστέων στους οποίους ενεργεί, όπως στην έκφραση `x + y`.
- Η σημειογραφία ***πρόθεσης*** ή ***προπορευόμενου τελεστή*** (prefix notation), όταν αυτός τοποθετείται πριν από τους τελεστέους, όπως στην έκφραση `+ x`.
- Η σημειογραφία ***παρελκόμενου τελεστή*** (postfix notation), όταν ο τελεστής τοποθετείται μετά από τους τελεστέους, όπως στην έκφραση `x +`.

Παρατηρήσεις:

1. Οι τελεστές είναι προτιμότερο να μην χρησιμοποιούνται σε μεικτούς τύπους. Για παράδειγμα, η έκφραση

```
out_int=my1_int+my2_int
```

δεν παρουσιάζει κανένα πρόβλημα, σε αντιδιαστολή με τον ακόλουθο μεικτό τύπο

```
out_float=my_double/my_int
```

2. Στη γλώσσα C υπάρχει διάκριση ανάμεσα στη διαίρεση ακεραίων και στη διαίρεση αριθμών κινητής υποδιαστολής. **Στη διαίρεση ακεραίων το αποτέλεσμα προέρχεται από το σχήμα (Διαιρετέος = Πηλίκο*Διαιρέτης + Υπόλοιπο)**, επομένως η διαίρεση του **5** με το **2** παράγει το ακέραιο πηλίκο **2** κι όχι το **2.5**. Για να ληφθεί ως αποτέλεσμα αριθμός κινητής υποδιαστολής, τουλάχιστον ένας από τους τελεστέους πρέπει να είναι αριθμός κινητής υποδιαστολής: η έκφραση `5.0/2` υπολογίζεται ως **2.5**.

2.5. Κατηγορίες εκφράσεων της γλώσσας C– προτεραιότητα και προσεταιριστικότητα

Οι εκφράσεις της γλώσσας C μπορούν να καταταγούν στις παρακάτω κατηγορίες:

- ***Σταθερές εκφράσεις***. Είναι εκφράσεις που περιέχουν μόνο σταθερές τιμές.
- ***Ακέραιες εκφράσεις και εκφράσεις κινητής υποδιαστολής***. Είναι εκφράσεις, οι οποίες μετά από όλες τις άμεσες και έμμεσες μετατροπές τύπων δίνουν αποτέλεσμα ακέραιου τύπου ή τύπου κινητής υποδιαστολής αντίστοιχα.
- ***Εκφράσεις δείκτη***. Είναι εκφράσεις με τιμή μία διεύθυνση. Περιλαμβάνουν μεταβλητές δείκτη (η έννοια του δείκτη θα αναλυθεί στο κεφάλαιο 6), τον τελεστή διεύθυνσης `&`, αλφαριθμητικές σταθερές και ονόματα πινάκων.

Ο υπολογισμός μίας έκφρασης δεν είναι πάντοτε απλή υπόθεση, ιδιαίτερα στην περίπτωση που υπάρχουν ένθετες (nested) εκφράσεις, δηλαδή εκφράσεις που είναι φωλιασμένες μέσα σε άλλες. Στην έκφραση

```
(((n+9)>=k) && j)
```

η έκφραση `n+9` είναι φωλιασμένη στην έκφραση `(n+9)>=k`, η οποία με τη σειρά της είναι φωλιασμένη στη συνολική έκφραση. Μία άλλη περίπτωση δυσχέρειας στον υπολογισμό είναι η διαδοχική παράθεση τελεστών: `4*9-12`, η οποία μπορεί να υπολογιστεί είτε ως `(4*9)-12=24` είτε ως `4*(9-12)=-12`, οδηγώντας σε διαφορετικά αποτελέσματα.

Για να αντιμετωπιστούν οι ανωτέρω δυσχέρειες έχει υιοθετηθεί μία σειρά εφαρμογής των τελεστών, η επονομαζόμενη **εφαρμοστική σειρά** (applicative order), η οποία στηρίζεται στις έννοιες της **προτεραιότητας** (precedence) και της **προσεταιριστικότητας** (associativity) των τελεστών.

Οι τελεστές ταξινομούνται σε επίπεδα προτεραιότητας, με τη σύμβαση ότι οι τελεστές υψηλότερου επιπέδου προτεραιότητας δρουν επί των τελεστέων πριν από τους τελεστές χαμηλότερου επιπέδου.

Η ύπαρξη περισσότερων τελεστών στο ίδιο επίπεδο προτεραιότητας επιβάλλει τον προσδιορισμό της **κατεύθυνσης εφαρμογής**, με την κατεύθυνση από τα αριστερά προς τα δεξιά να είναι ευρύτερα χρησιμοποιούμενη. Ένας τελεστής καλείται **αριστερής προσεταιριστικότητας** (left associative), όταν σε εκφράσεις που περιέχουν πολλά στιγμιότυπα του τελεστή η ομαδοποίηση γίνεται από τα αριστερά προς τα δεξιά. Έτσι, η έκφραση `9-3-2` υπολογίζεται ως `(9-3)-2`. Οι τελεστές `+`, `-` ,`*`, `/` είναι όλοι αριστερής προσεταιριστικότητας.

Για τη γλώσσα C παράδειγμα **δεξιάς προσεταιριστικότητας** αποτελεί η ύψωση σε δύναμη και ο τελεστής ανάθεσης (`=`). Στην έκφραση `num1=num2=10` εφαρμόζεται πρώτα ο δεξιός τελεστής ανάθεσης, με αποτέλεσμα η `num2` να αποκτήσει την τιμή **10**. Ακολούθως, εφαρμόζεται ο αριστερός τελεστής ανάθεσης, έτσι ώστε η `num1` εξισώνεται με τη `num2` και αποκτά την τιμή **10**.

Η προτεραιότητα και το είδος προσεταιριστικότητας των τελεστών παρατίθενται στον **Πίνακα 2.7**, όπου οι τελεστές έχουν τεθεί με *σειρά φθίνουσας προτεραιότητας*:

Τελεστές	Είδος προσεταιριστικότητας
`()` `[]` `->`	από αριστερά προς τα δεξιά
`!` `~` `++` `--` `+` `-` `*` `&`	από δεξιά προς τα αριστερά

(τύπος) sizeof	
* / % (αριθμητικοί τελεστές)	από αριστερά προς τα δεξιά
+ - (αριθμητικοί τελεστές)	»
<< >>	»
< <= > >=	»
== !=	»
&	»
^	»
|	»
&&	»
||	»
?:	από δεξιά προς τα αριστερά
= += -= *= %= &= ^= |= <<= >>=	»
	από αριστερά προς τα δεξιά

Πίνακας 2.7 *Προτεραιότητα και προσεταιριστικότητα τελεστών*

Με βάση τα προαναφερθέντα, είναι προφανές ότι με τους κανόνες προτεραιότητας και προσεταιριστικότητας δεν είναι πάντοτε απαραίτητη η χρήση παρενθέσεων για τον προσδιορισμό του τρόπου υπολογισμού της τιμής των εκφράσεων. Ωστόσο, οι παρενθέσεις χρησιμοποιούνται για τους ακόλουθους λόγους:

- Για να προσδιοριστεί συγκεκριμένη σειρά εφαρμογής, όπως στην έκφραση `(2-3)*4`.
- Για να καταστεί μία έκφραση ευανάγνωστη, όπως στην έκφραση `2-(3*4)`, παρά το γεγονός ότι στην τελευταία περίπτωση αποτελεί πλεονασμό.

Στην περίπτωση ένθετων παρενθέσεων ο μεταγλωττιστής εφαρμόζει πρώτα τις εσωτερικές παρενθέσεις. Για παρενθέσεις, όμως, που βρίσκονται στο ίδιο βάθος ένθεσης, δεν ορίζεται η σειρά υπολογισμού.

2.5.1. Παράδειγμα

Με χρήση του **Πίνακα 2.7** να υπολογιστούν οι εκφράσεις:
(α) `x=17-2*8`
(β) `y=17-2-8`

(α) Βάσει της προτεραιότητας, πρώτα θα εκτελεστεί ο πολλαπλασιασμός `2*8` και το γινόμενο που θα προκύψει θα αποτελέσει τελεστή στην αφαίρεση από το `17`. Το τελικό αποτέλεσμα `1` ανατίθεται στη μεταβλητή `x` που βρίσκεται αριστερά του τελεστή ανάθεσης `=`.

`x=17-(2*8)=17-16=1`

(β) Εφόσον εμφανίζονται δύο στιγμιότυπα του τελεστή `-`, οι κανόνες προσεταιριστικότητας καθορίζουν την εκτέλεση των πράξεων από τα αριστερά προς τα δεξιά:

`y=(17-2)-8=15-8=7`

2.6. Τελεστές μοναδιαίας αύξησης και μείωσης

Ο τελεστής μοναδιαίας αύξησης (increment operator) συμβολίζεται `++`. Με χρήση αυτού του τελεστή η έκφραση `num=num+1` είναι ισοδύναμη με την έκφραση `num++`.

Αντίστοιχα, ο τελεστής μοναδιαίας μείωσης (decrement operator) συμβολίζεται `--` και η έκφραση `num=num-1` είναι ισοδύναμη με την έκφραση `num--`.

2.6.1. Παράδειγμα

Προπορευόμενοι και παρελκόμενοι τελεστές μοναδιαίας αύξησης και μείωσης: να υπολογιστούν οι τιμές των **x** και **y** στις προτάσεις του **Πίνακα 2.8**, οι οποίες εκτελούνται διαδοχικά.

Πρόταση	Τιμή x	Τιμή y
`int x=10, y=15;`	10	15
`++x;`	11	15
`y=--x;`	10	5
`y=x-- + y;`	9	15
`y=y - x--;`	8	6

Πίνακας 2.8

2.6.2. Παράδειγμα

Να προσδιοριστεί η τιμή των **x**, **y** και **z** μετά την εκτέλεση καθεμιάς από τις παρακάτω προτάσεις, θεωρώντας ότι πριν την εκτέλεση της κάθε πρότασης οι τιμές των **x** και **y** είναι το **10** και το **15** αντίστοιχα.

(α) `z=++x + y;`

(β) `z=--x + y;`

(γ) `z=x++ + y;`

(δ) `z=x-- + y;`

(ε) `z=x-- + ++y;`

(στ) `z=++x + y--;`

Στην περίπτωση του προπορευόμενου τελεστή, το σύστημα πρώτα εκτελεί την αύξηση ή μείωση και μετά χρησιμοποιεί τη νέα τιμή της μεταβλητής στον υπολογισμό της τιμής της έκφρασης (προτάσεις *(α)* και *(β)*). Αντίθετα, στην περίπτωση του παρελκόμενου τελεστή το σύστημα πρώτα χρησιμοποιεί την τιμή της μεταβλητής για τον υπολογισμό της τιμής της έκφρασης και μετά εκτελεί την αύξηση ή μείωση της τιμής της μεταβλητής (προτάσεις *(γ)* και *(δ)*). Στις προτάσεις *(ε)* και *(στ)* εμφανίζονται και προπορευόμενος και παρελκόμενος τελεστής, οπότε το σύστημα χειρίζεται την κάθε περίπτωση ξεχωριστά, σύμφωνα με τους προαναφερθέντες κανόνες. Τα αποτελέσματα παρατίθενται στον **Πίνακα 2.9**:

Πρόταση	Τιμή x	Τιμή y	Τιμή z
`z=++x + y;`	11	15	26
`z=--x + y;`	9	15	24
`z=x++ + y;`	11	15	25
`z=x-- + y;`	9	15	25
`z=x-- + ++y;`	9	16	26
`z=++x + y--;`	11	14	26

Πίνακας 2.9

2.7. Τελεστές ανάθεσης

Οι τελεστές ανάθεσης (assignment operators) εκτελούν μία πράξη ανάμεσα στους τελεστέους και εκχωρούν το αποτέλεσμα σε έναν από τους τελεστέους:

- `x*=100;` Εκτελεί την πράξη του πολλαπλασιασμού μεταξύ των `x` και **100** και εκχωρεί το αποτέλεσμα στον τελεστέο `x`. Αντιστοιχεί στην πρόταση `x=x*100;`
- `x*=y+12;` Αντιστοιχεί στην πρόταση `x=x*(y+12);` κι όχι στην πρόταση `x=x*y+12;`

Τελεστές ανάθεσης δημιουργούν και οι τελεστές διαχείρισης δυαδικών ψηφίων. Οι τελεστές αυτοί είναι: `>>=`, `<<=`, `&=`, `^=`, και `|=` (βλ. §2.10).

Οι τελεστές ανάθεσης σε συνδυασμό με τους τελεστές μοναδιαίας αύξησης/μείωσης γίνονται αιτία δημιουργίας παρενεργειών, για τον λόγο αυτό αναφέρονται και ως **παρενεργοί τελεστές** (side–effect operators). Οι παρενέργειες αυτές έχουν ως αποτέλεσμα την απροσδιόριστη συμπεριφορά του συστήματος ως προς τον τρόπο υπολογισμού της τιμής της μεταβλητής `x` σε εκφράσεις, όπως `x= ++x + 4;`

2.8. Συσχετιστικοί τελεστές

Οι συσχετιστικοί τελεστές (relational operators) συγκρίνουν δύο τελεστέους. Οι βασικοί τελεστές της κατηγορίας αυτής παρατίθενται στον **Πίνακα 2.10**:

Τελεστής	Δράση
`<`	μικρότερο από
`>`	μεγαλύτερο από
`<=`	μικρότερο ή ίσον από
`>=`	μεγαλύτερο ή ίσον από
`==`	ίσο
`!=`	διάφορο

Πίνακας 2.10 *Σύμβολα συσχετιστικών τελεστών*

Το αποτέλεσμα της χρήσης των συσχετιστικών τελεστών είναι είτε **ΑΛΗΘΕΣ** (true) είτε **ΨΕΥΔΕΣ** (false). Για παράδειγμα, η τιμή της έκφρασης `(3<2)` είναι ψευδής ενώ η τιμή της έκφρασης `(2==2)` είναι αληθής.

Στη γλώσσα C (και σε πολλές άλλες γλώσσες προγραμματισμού) η τιμή **ΑΛΗΘΗΣ** αντιστοιχεί στον ακέραιο **1** και η τιμή **ΨΕΥΔΗΣ** αντιστοιχεί στον ακέραιο **0**.

Παρατήρηση: Συγκρίνοντας τους αριθμητικούς με τους συσχετιστικούς τελεστές προκύπτει ότι και οι δύο χρησιμοποιούν αριθμητικούς τελεστέους, π.χ. `(num+10)` και `(num<10)`, όπου `num` είναι μία ακέραια μεταβλητή. Ωστόσο, οι αριθμητικοί τελεστές μπορούν να δώσουν ως αποτέλεσμα οποιονδήποτε αριθμό (για κάθε τιμή του `num` η πρόταση `(num+10)` δίνει μία άλλη τιμή), ενώ οι συσχετιστικοί τελεστές έχουν δίτιμη έξοδο (για κάθε τιμή του `num` μικρότερη του **10** η πρόταση `(num<10)` δίνει **TRUE** (1) και για όλες τις άλλες τιμές του `num` δίνει **FALSE** (0)).

2.9. Λογικοί τελεστές

Οι λογικοί τελεστές δρουν επί ενός ή δύο τελεστέων και λειτουργούν με βάση τη δίτιμη άλγεβρα Boole. Τόσο οι είσοδοι όσο και οι έξοδοι μπορούν να λάβουν μόνο δύο τιμές, **TRUE** και **FALSE**. Οι τελεστές της κατηγορίας αυτής παρατίθενται στον **Πίνακα 2.11**, ενώ στον **Πίνακα 2.12** περιγράφεται ο τρόπος λειτουργίας τους (πίνακας αληθείας):

Τελεστής	Δράση
`&&`	λογικό AND

`||`	λογικό OR
`!`	λογικό NOT

Πίνακας 2.11 *Σύμβολα λογικών τελεστών*

x	y	x && y	x || y	!x
True	True	True	True	False
True	False	False	True	
False	True	False	True	True
False	False	False	False	

Πίνακας 2.12 *Πίνακας αληθείας των λογικών τελεστών*

2.9.1. Παράδειγμα

Για `x=16` και `y=-6` να υπολογιστούν οι εκφράσεις:

(α) `(x+9)<(13-y)`

(β) `(x<5) || (y>10)`

(γ) `(x>=7) && (!((x-15)<y))`

Αντικαθιστώντας τις αριθμητικές τιμές προκύπτει:

(α) `(16+9)<(13-(-6))` → `25<19` → **FALSE**

(β) `(16<5) || (-6>10)` → `FALSE || FALSE` → **FALSE**

(γ) `(16>=7) && (!((16-15)<(-6)))` → `TRUE && (!(1<(-6)))` → `TRUE && (!FALSE)` → `TRUE && TRUE` → **TRUE**

2.9.2. Παράδειγμα

Να εξαχθεί η έξοδος του ακόλουθου προγράμματος:

```
#include <stdio.h>

int main()
{
   int x1=4,x2=-17,x3=10,x4;
   /* 1η ομάδα προτάσεων */
   x3=++x1 - --x2;
   printf("x1=%d, x2=%d, x3=%d\n",x1,x2,x3);
   /* 2η ομάδα προτάσεων */
   x4=((x1<1000) || !((x2-x3)>(x1=x1+8)));
   printf("x1=%d, x4=%d, %d\n",x1,x4,((x1>3) && !(x4/x2)));

   return 0;
}
```

```
x1=5, x2=-18, x3=23
x1=5, x4=1, 1
```

Εικόνα 2.8 *Η έξοδος του προγράμματος του παραδείγματος 2.9.2*

- Στην πρώτη ομάδα προτάσεων, λόγω των προπορευόμενων τελεστών η αρχική τιμή της **x1** αυξάνεται κατά **1** και της **x2** μειώνεται κατά **1**, με αποτέλσμα οι νέες τιμές τους να είναι **5** και **-18**. Οι τιμές αυτές αφαιρούνται μεταξύ τους και το αποτέλεσμα (**5 - (-18) = 23**) εκχωρείται στη μεταβλητή **x3**.
- Στη δεύτερη ομάδα προτάσεων οι έλεγχοι οδηγούν στα ακόλουθα:
  - **x1<1000** → **5<1000** → **TRUE**
  - **(x2-x3)>(x1=x1+8))** → **(-18-23)>(x1=5+8))** → **-41>13** → **FALSE**
  - **!((x2-x3)>(x1=x1+8))** → **!FALSE** → **TRUE**
  - **x4=TRUE || TRUE** → **TRUE**, δηλαδή **1**
- Ο τελευταίος προσδιοριστής στην **printf()** της δεύτερης ομάδας προτάσεων αντιστοιχεί στο αποτέλεσμα του ελέγχου: **((x1>3) && !(x4/x2))** → **((5>3) && !(1/23))** → **TRUE && !0** → **TRUE && !FALSE** → **TRUE && TRUE** → **TRUE**, δηλαδή **1**

2.10. Τελεστές διαχείρισης δυαδικών ψηφίων

Στη γλώσσα C υπάρχουν έξι τελεστές διαχείρισης δυαδικών ψηφίων (bitwise operators), οι οποίοι επιτρέπουν την εκτέλεση πράξεων σε επίπεδο bit. Οι τελεστές αυτοί βρίσκουν εφαρμογή στα πεδία της κωδικοποίησης και της συμπίεσης δεδομένων. Οι τελεστές της κατηγορίας αυτής παρατίθενται στον **Πίνακα 2.13**, ενώ στον **Πίνακα 2.14** περιγράφεται ο τρόπος λειτουργίας τους (πίνακας αληθείας):

Τελεστής	Δράση
&	λογικό AND
|	λογικό OR
~	συμπλήρωμα ως προς 1
^	eXclusive OR
<<	ολίσθηση προς τα αριστερά
>>	ολίσθηση προς τα δεξιά

Πίνακας 2.13 *Σύμβολα τελεστών διαχείρισης δυαδικών ψηφίων*

x	y	x & y	x | y	x ^ y	~x
1	1	1	1	0	0
1	0	0	1	1	
0	1	0	1	1	1
0	0	0	0	0	

Πίνακας 2.14 *Πίνακας αληθείας των τελεστών διαχείρισης δυαδικών ψηφίων*

- Η ολίσθηση προς τα αριστερά (bit left shift) μετατοπίζει όλα τα bits του αριστερού τελεστέου κατά τόσες θέσεις προς τα αριστερά, όση είναι η τιμή του δεξιού τελεστέου. Τα bits που απομένουν άνευ περιεχομένου στο δεξί τμήμα του τελεσταίου, συμπληρώνονται με μηδενικά. Η ολίσθηση αυτή πολλαπλασιάζει την υφιστάμενη τιμή του αριστερού τελεστέου επί 2^n, όπου **n** είναι ο αριθμός των θέσεων ολίσθησης.
- Η ολίσθηση προς τα δεξιά (bit right shift) μετατοπίζει όλα τα bits του αριστερού τελεστέου κατά τόσες θέσεις προς τα δεξιά, όση είναι η τιμή του δεξιού τελεστέου. Τα bits που απομένουν άνευ περιεχομένου στο αριστερό τμήμα του τελεσταίου, συμπληρώνονται με μηδενικά. Η ολίσθηση αυτή διαιρεί την υφιστάμενη τιμή του αριστερού τελεστέου διά 2^n, όπου **n** είναι ο αριθμός των θέσεων ολίσθησης.
- Όταν χρησιμοποιούνται τελεστές ολίσθησης, είναι προτιμότερο να εφαρμόζεται σε μη προσημασμένες μεταβλητές, καθώς η εφαρμογή τους σε αρνητικούς αριθμούς εξαρτάται από τον εκάστοτε μεταγλωττιστή.

2.10.1. Παράδειγμα

Να υπολογιστούν οι εκφράσεις:
(α) `10101010 & 00001111`

(β) `010101010 | 11110000`

(γ) `11001100 ^ 00111100`

(δ) `~11010110`

(ε) `9<<4`

(στ) `36>>2`

(α) `10101010 & 00001111` → `(1&0)(0&0)(1&0)(0&0)(1&1)(0&1)(1&1)(0&1)` → `00001010`

(β) `01010101 | 11110000` → `(0|1)(1|1)(0|1)(1|1)(0|0)(1|0)(0|0)(1|0)` → `11110101`

(γ) `11001100 ^ 00111100` → `(1^0)(1^0)(0^1)(0^1)(1^1)(1^1)(0^0)(0^0)` → `11110000`

(δ) `~11010110` → `00101001`

(ε) Ο δεκαδικός αριθμός `9` αντιστοιχεί στον δυαδικό αριθμό `00001001` → $b_7b_6b_5b_4b_3b_2b_1b_0$. Μετακίνηση κατά 4 θέσεις προς τα αριστερά σημαίνει ότι θα γίνουν οι ακόλουθες αντικαταστάσεις: $b_7=b_3$, $b_6=b_2$, $b_5=b_1$, $b_4=b_0$, $b_3=b_2=b_1=b_0=0$ και ο αριστερός τελεσταίος γίνεται `10010000`, ο οποίος αντιστοιχεί στον δεκαδικό αριθμό `144`.

(στ) Ο δεκαδικός αριθμός `36` αντιστοιχεί στον δυαδικό αριθμό `00100100` → $b_7b_6b_5b_4b_3b_2b_1b_0$. Μετακίνηση κατά 2 θέσεις προς τα δεξιά σημαίνει ότι θα γίνουν οι ακόλουθες αντικαταστάσεις: $b_0=b_2$, $b_1=b_3$, $b_2=b_4$, $b_3=b_5$, $b_4=b_6$, $b_5=b_7$, $b_6=b_7=0$ και ο αριστερός τελεσταίος γίνεται `00001001`, ο οποίος αντιστοιχεί στον δεκαδικό αριθμό `9`.

2.11. Μετατροπές τύπων

Όταν ένας τελεστής έχει τελεστέους διαφορετικών τύπων δεδομένων, αυτοί μετατρέπονται σε ενιαίο τύπο. Η μετατροπή είτε γίνεται αυτόματα από τον υπολογιστή, οπότε καλείται **έμμεση** (implicit conversion), είτε άμεσα από τον προγραμματιστή, οπότε καλείται **ρητή** (explicit conversion).

2.11.1. Έμμεσες μετατροπές

Οι έμμεσες μετατροπές διευκολύνουν την εργασία του προγραμματιστή, ο οποίος όμως θα πρέπει σε κάθε περίπτωση να γνωρίζει τις συνέπειες μίας μετατροπής. Για παράδειγμα, η έκφραση `3.0+1/2` δεν δίνει τιμή **3.5**, όπως πιθανόν να ήταν αναμενόμενο, αλλά **3.0**.

Η διαδικασία της αυτόματης μετατροπής στηρίζεται στους ακόλουθους κανόνες:

- Σε κάθε πράξη που υπάρχουν δύο τύποι δεδομένων, ο τύπος χαμηλότερης ιεραρχίας μετατρέπεται στον υψηλότερης ιεραρχίας χωρίς να υπάρχει απώλεια πληροφορίας.
- Οι τύποι της γλώσσας κατατάσσονται ιεραρχικά ανάλογα με το μέγεθος της μνήμης που απαιτούν για αποθήκευση, όπως παρακάτω:

`char < int < long < float < double`

Ο τύπος `unsigned` ακολουθεί τον αντίστοιχο προσημασμένο τύπο.

- Όλοι οι μεταγλωττιστές της C, όταν υπολογίζουν αριθμητικές εκφράσεις, μετατρέπουν αυτόματα τον τύπο **char** σε **int** και τον **float** σε **double**.

2.11.1.1. Παράδειγμα

Να αναλυθεί η λειτουργία του ακόλουθου προγράμματος:

```
#include <stdio.h>
int main()
{
   char ch;
   int i;
   float fl;
   fl=i=ch='A'; // (1)
   printf( "ch=%c, i=%d, fl=%2.2f,\n",ch,i,fl );
   ch=ch+1;     // (2)
   i=fl+2*ch;   // (3)
   fl=2.0*ch+i;   // (4)
   printf( "ch=%c, i=%d, fl=%2.2f,\n",ch,i,fl );

   return 0;
}
```

```
ch=A, i=65, fl=65.00,
ch=B, i=197, fl=329.00
```

Εικόνα 2.8 *Η έξοδος του προγράμματος του παραδείγματος 2.11.1.1*

(1) Ο χαρακτήρας **'A'** αποθηκεύεται ως χαρακτήρας στη μεταβλητή **ch**. Η μεταβλητή **i** λαμβάνει την τιμή του ακέραιου από τη μετατροπή του **'A'** (**65**), ενώ η μεταβλητή **fl** λαμβάνει την τιμή του αριθμού κινητής υποδιαστολής που προέρχεται από το **65** (**65.00**).

(2) Η πρόσθεση της μονάδας γίνεται στην ακέραια τιμή του **'A'**. Το αποτέλεσμα, **66,** αντιστοιχεί στον χαρακτήρα **'B'**, ο οποίος αποθηκεύεται στη μεταβλητή **ch**.

(3) Η πράξη δίνει **2*66+65.00=197.00**. Το αποτέλεσμα μετατρέπεται σε **int**, **197**, και αποθηκεύεται στη μεταβλητή **i**.

(4) Η πράξη δίνει **2.0*66+197=329.00** (οι αριθμοί **int** μετατρέπονται σε **float**). Το αποτέλεσμα αποθηκεύεται στη μεταβλητή **fl**.

2.11.2. Ρητές μετατροπές– τελεστής typecast

Εκτός από τις αυτόματες μετατροπές, η C επιτρέπει ρητές μετατροπές μίας τιμής σε έναν διαφορετικό τύπο δεδομένων. Η διαδικασία ονομάζεται **προσαρμογή** ή **εκμαγείο** (casting) και ο τελεστής μετατροπής τύπου ή **cast** τελεστής, όπως αποκαλείται, είναι μοναδιαίος κι έχει τη μορφή **(τύπος δεδομένων)**, π.χ. **(float)**. Τοποθετείται μπροστά από μία έκφραση, για να μετατρέψει την τιμή της στον περικλειόμενο σε παρενθέσεις τύπο δεδομένων:

```
int i,j;
float f1,f2,f3;
i=5;
j=2;
fl=i/j+0.5;        /* Αποτέλεσμα: 2.5 */
```

```
f2=(float)i/(float)j+0.5; /* Αποτέλεσμα: 3.0 */
f3=i/j+0.5;       /* Αποτέλεσμα: 2.5 */
```

2.12. Ο τελεστής sizeof

Ο τελεστής **sizeof** είναι μοναδιαίος και δρα είτε σε σε μεταβλητή ή ολόκληρη έκφραση, π.χ. **sizeof(x+y)** είτε σε τύπο δεδομένων, πχ. **sizeof(int)**.

Σε κάθε περίπτωση επιστρέφει τον αριθμό των bytes που η τιμή της έκφρασης ή ο τύπος των δεδομένων καταλαμβάνει στη μνήμη. Προσοχή θα πρέπει να δοθεί στο γεγονός ότι το σύστημα δεν υπολογίζει την τιμή της έκφρασης κι έτσι πιθανή ύπαρξη παρενεργειών από τους τελεστές δεν δημιουργεί παρενέργειες στη λειτουργία του **sizeof**.

Ερωτήσεις αυτοαξιολόγησης- ασκήσεις

Ερωτήσεις αυτοαξιολόγησης

Ο αναγνώστης καλείται να επιλέξει μία από τις τέσσερις απαντήσεις.

(1) Η ακολουθία διαφυγής **'\n'** δηλώνει:

(α) Εκτύπωση του χαρακτήρα **'\'**

(β) Εκτύπωση του χαρακτήρα **'n'**

(γ) Μετακίνηση του κέρσορα στην επόμενη γραμμή

(δ) Μετακίνηση του κέρσορα προς τα δεξιά κατά μία θέση στηλογνώμονα

(2) Η μορφοποιούμενη συνάρτηση εκτύπωσης στο κανάλι εξόδου (οθόνη) ονομάζεται:

(α) **print**

(β) **write**

(γ) **fprint**

(δ) **printf**

(3) Ποιο από τα ακόλουθα ονόματα μεταβλητών είναι σωστό;

(α) **_temp_in_F**

(β) **total%**

(γ) **$product**

(δ) **3rd**

(4) Να υπολογιστούν οι τελικές τιμές των **x** και **y** στις ακόλουθες εκφράσεις που εκτελούνται διαδοχικά.

```
int x=10, y=20;
++ x;
y=--x;
y=x-- + y;
y=y - x++;
```

(α) **x=9**, **y=9**

(β) **x=9**, **y=11**

(γ) **x=10**, **y=10**

(δ) **x=10**, **y=11**

(5) Ποια είναι τα αποτελέσματα του ακόλουθου προγράμματος;

```
#include <stdio.h>
int main()
```

```
{
   char ch;
   int i;
   float fl;
   fl='A';
   i=ch='B'+2;
   printf( "ch=%c, i=%d, fl=%6.3f\n",ch,i,fl );
   ch=ch+1;
   i=(int)(2.5*fl);
   printf( "ch=%c, i=%d\n",ch,i );

   return 0;
}
```

(α) `ch=D, i=68, fl=65.000`
`ch=E, i=162`
(β) `ch=D, i=65, fl=65.00`
`ch=D, i=65`
(γ) `ch=D, i=D, fl=65.000`
`ch=E, i=162`
(δ) `ch=65, i=68, fl=65.00`
`ch=E, i=162`

Ασκήσεις

Άσκηση 1

Να γραφεί πρόγραμμα, το οποίο θα δέχεται από το πληκτρολόγιο τα μήκη των πλευρών ενός ορθογώνιου τριγώνου και θα τυπώνει την υποτείνουσά του (για τον υπολογισμό της τετραγωνικής ρίζας του αριθμού `x` να χρησιμοποιηθεί η συνάρτηση `sqrt(x)`, η οποία δηλώνεται στο αρχείο κεφαλίδας `math.h`).

Άσκηση 2

Να γραφεί πρόγραμμα, το οποίο θα δέχεται από το πληκτρολόγιο έναν ακέραιο αριθμό `x`, που εκφράζει δευτερόλεπτα, και θα εμφανίζει στην οθόνη τις ώρες, λεπτά και δευτερόλεπτα που αντιστοιχούν στον `x`. Σημειώνεται ότι ο δυαδικός τελεστής `%` υπολογίζει το υπόλοιπο της διαίρεσης ακεραίων.

Άσκηση 3

Να υπολογιστούν οι παρακάτω εκφράσεις τελεστών διαχείρισης δυαδικών ψηφίων:

(α) `01011101 & 01001111`
(β) `01010101 | 10110100`
(γ) `11000011 ^ 01101100`
(δ) `~10111100`
(ε) `8<<3`
(στ) `48>>2`

Άσκηση 4

Να γραφεί πρόγραμμα, το οποίο:
***(α)* Θα αποθηκεύει σε μεταβλητές τα ακόλουθα:**

- **Την ημέρα γεννήσής σας (π.χ. 12)**
- **Τον μήνα γεννήσής σας (π.χ. 7)**
- **Το έτος γέννησής σας (π.χ. 1996)**
- **Το ύψος σας σε μέτρα (π.χ. 1.85)**
- **Το πρώτο γράμμα του ονόματός σας (π.χ. X)**

***(β)* Τα ανωτέρω στοιχεία θα εμφανίζονται στην οθόνη. Η ημερομηνία θα εμφανίζεται στη μορφή ημέρα/μήνας/έτος. Το ύψος θα εμφανίζεται με δύο δεκαδικά ψηφία.**

Άσκηση 5

Να γραφεί πρόγραμμα, το οποίο:

(α) **Θα διαβάζει από το πληκτρολόγιο δύο ακέραιους αριθμούς, τους οποίους θα αποθηκεύει σε μεταβλητές και θα τους εμφανίζει στην οθόνη.**

(β) **Θα υπολογίζει το άθροισμα και το γινόμενο των δύο αριθμών και θα τα εμφανίζει στην οθόνη.**

(γ) **Θα εμφανίζει στην οθόνη τις απόλυτες τιμές όλων των ανωτέρω αριθμών (η συνάρτηση `abs(x)`, η οποία βρίσκεται στο αρχείο κεφαλίδας `math.h`, επιστρέφει την απόλυτη τιμή ενός ακέραιου αριθμού `x`).**

Άσκηση 6

Να γραφεί πρόγραμμα, το οποίο:

(α) **Θα διαβάζει από το πληκτρολόγιο δύο φυσικούς αριθμούς, τους οποίους θα αποθηκεύει σε μεταβλητές και θα τους εμφανίζει στην οθόνη.**

(β) **Θα υπολογίζει την περίμετρο του ορθογώνιου παραλληλογράμμου που δημιουργείται, όταν τεθούν ως μήκη πλευρών οι ανωτέρω ακέραιοι αριθμοί. Η περίμετρος θα εμφανίζεται στην οθόνη.**

Άσκηση 7

Να εξαχθούν τα αποτελέσματα του παρακάτω προγράμματος:

```
int main()
{
  int x,y,z;

  x=0;
  y=10;
  printf( "\t\n\nx=%d y=%d\n",x,y );

  z=(x>y);
  printf( "x > y is %d\n",z );

  z=(x==y);
  printf( "x==y is %d\n",z );

  z=(x!=y);
  printf( "x != y is %d\n",z );

  z=(x && y);
  printf( "x && y is %d\n",z );

  z=!(x && y) || (x || y);
  printf( "!(x && y) || (x || y) is %d\n",z );

  x = 10;
  printf( "\t\n\n\nx=%d y=%d\n",x,y );

  z=(x>y);
  printf( "x > y is %d\n",z );

  z=(x!=y);
  printf( "x != y is %d\n",z );

  z=(x && y);
  printf( "x && y is %d\n",z );
```

```
    float f1, f2;
    f1=5/(float)x;
    f2=5.0/(float)x;
    printf( "5.0/(float)%d = %f\n",x,f2 );
    printf( "x=%d NOT(x)=%d\n",x,!x );

    z=10+x;
    printf( "10+%d = %d\n",x,z );

    z=10+(!x);
    printf( "10+NOT(%d) = %d\n",x,z );

    return 0;
}
```

Βιβλιογραφία κεφαλαίου

Θραμπουλίδης, Κ. (2002), *Διαδικαστικός Προγραμματισμός - C (Τόμος Α)*, 2η έκδοση, Εκδόσεις Τζιόλα.

Καρολίδης, Δ. (2013), *Μαθαίνετε Εύκολα C*, αυτοέκδοση.

Τσελίκης, Γ. & Τσελίκας, Ν. (2012), *C από τη Θεωρία στην Εφαρμογή*, 2η έκδοση.

Χατζηγιαννάκης, Ν. (2012), *Η Γλώσσα C σε Βάθος*, 4η Έκδοση, Εκδόσεις Κλειδάριθμος.

Deitel, H. & Deitel, P. (2014), *C Προγραμματισμός*, 7η έκδοση, Εκδόσεις Γκιούρδα.

Horton, I. (2006), *Beginning C – from Novice to Professional*, 4th ed., Apress.

Roberts, E. (2008), *Η Τέχνη και Επιστήμη της C*, Εκδόσεις Κλειδάριθμος.