Προγραμματισμός Η/Υ Ι

02.
Αναπαράσταση Πληροφορίας, Δεδομένα, Τύποι και Τιμές

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2019-2020 | ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ Τ.Ε.

Αναπαράσταση Πληροφορίας

Δυαδικά ψηφία (binary digits, bits)

- οι υπολογιστές είναι σε θέση να αναγνωρίζουν 2 διακριτές καταστάσεις:
  1. περνά ρεύμα
  2. δεν περνά ρεύμα

το δυαδικό ψηφίο (binary digit ή bit)

- είναι η μικρότερη μονάδα μέτρησης
- εκφράζει αυτές τις 2 διακριτές καταστάσεις
  - τιμή **`1`** → περνά ρεύμα
  - τιμή **`0`** → δεν περνά ρεύμα

Αναπαράσταση Πληροφορίας
Κωδικοποίηση

- όλα τα στοιχεία ενός πληροφοριακού συστήματος κωδικοποιούνται σε bits
  - οι εντολές που εκτελούνται
  - τα δεδομένα (π.χ. κείμενο, αριθμοί, αναλογικό σήμα, εικόνες και video)

επομένως, η κωδικοποίηση των δεδομένων:

- βασίζεται στο δυαδικό σύστημα και χρησιμοποιεί δυαδικούς αριθμούς (→ δυαδική αναπαράσταση)
- συνιστά τρόπο αναπαράστασης των δεδομένων στους υπολογιστές (ψηφιακά δεδομένα)
- λαμβάνει υπόψιν της το είδος των δεδομένων που πρόκειται να αναπαρασταθούν
- αφιερώνει συγκεκριμένο αριθμό από bits για κάθε στοιχείο των δεδομένων που πρόκειται να αναπαραστήσει

Κωδικοποίηση

1ο Παράδειγμα

- ? πόσα bits απαιτούνται για τα σημεία του ορίζοντα;
  - ✍ βορράς, νότος, ανατολή, δύση
  - ▶ απαιτούνται **4** διακριτές καταστάσεις
  - ❖ με 1 bit
    - ▶ 1η κατάσταση: 0
    - ▶ 2η κατάσταση: 1
    - ✖ μπορούν να αναπαραστηθούν **2** καταστάσεις συνολικά → άρα, το **1** bit **δεν** επαρκεί!
  - ❖ με 2 bits
    - ▶ 1η κατάσταση: 00
    - ▶ 2η κατάσταση: 01
    - ▶ 3η κατάσταση: 10
    - ▶ 4η κατάσταση: 11
    - ✓ μπορούν να αναπαραστηθούν **4** καταστάσεις συνολικά → άρα, τα **2** bits **επαρκούν**!

Σημείο	Κωδικοποίηση (δυαδικός αριθμός)
βορράς	**00**
νότος	**01**
ανατολή	**10**
δύση	**11**

Κωδικοποίηση

2ο Παράδειγμα

- ? πόσα bits απαιτούνται για τα 24 κεφαλαία γράμματα του ελληνικού αλφάβητου;
  - απαιτούνται **24** διακριτές καταστάσεις
  - × με **1** bit → έχουμε **2** διαφορετικές καταστάσεις – **δεν** επαρκούν!
  - × με **2** bits → έχουμε **4** διαφορετικές καταστάσεις – **δεν** επαρκούν!
  - × με **3** bits → έχουμε **8** διαφορετικές καταστάσεις – **δεν** επαρκούν!
  - × με **4** bits → έχουμε **16** διαφορετικές καταστάσεις – **δεν** επαρκούν!
  - ✓ με **5** bits → έχουμε **32** διαφορετικές καταστάσεις – **επαρκούν**
    - μάλιστα, περισσεύουν 8 καταστάσεις
  - ✍ κάθε κωδική ακολουθία πρέπει να περιγράφει μόνο ένα γράμμα
  - ✍ η επιλογή της κωδικής ακολουθίας (από bits) κάθε γράμματος δεν είναι μοναδική, π.χ. αντί για 00000, η κωδικοποίηση του 'Α' θα μπορούσε να είναι: 10000 ή 01000 ή ...

Γράμμα	Κωδικοποίηση (δυαδικός αριθμός)
Α	00000
Β	00001
Γ	00010
Δ	00011
...	...
Ψ	10110
Ω	10111

Κωδικοποίηση

- ? πόσα bits απαιτούνται;
  - ✍ με k bits μπορούμε ν' αναπαραστήσουμε $N=2^k$ διαφορετικές καταστάσεις

    (και τότε προφανώς ισχύει $k=\log_2 N$)

- υπάρχουν πολλές διαφορετικές κωδικοποιήσεις για τα ίδια στοιχεία, αρκεί σε κάθε κωδικοποίηση:
  - μία κωδική ακολουθία να περιγράφει ένα (και μόνο ένα) στοιχείο των δεδομένων
  - (ισοδύναμα) κάθε στοιχείο να έχει μία και μόνο μία κωδική ακολουθία
    - αυτό δεν ισχύει πάντα

- για κάθε τύπο δεδομένων έχουν προταθεί πρότυπα κωδικοποίησης
  - για χαρακτήρες: ASCII, ISO και UNICODE
  - για την αναπαράσταση προσημασμένων (θετικών ή αρνητικών) αριθμών: πρόσημο και μέτρο, συμπλήρωμα ως προς 1, συμπλήρωμα ως προς 2

Αναπαράσταση Κειμένου

- αφορά:
  - χαρακτήρες που μπορούν να εκτυπωθούν:
    `A-Z, a-z, 0-9, !, @, #, $, %, ^, &, *, (, ), _, -, +, /`
  - χαρακτήρες ελέγχου (που δεν εκτυπώνονται):
    `αλλαγή γραμμής, <enter>, <backspace>`
- σε κάθε κωδικοποίηση χρησιμοποιείται ένας συγκεκριμένος αριθμός από bits
  - ανάλογα με τον αριθμό των χαρακτήρων που πρόκειται να αναπαρασταθούν
- σε κάθε χαρακτήρα αντιστοιχεί ένας συγκεκριμένος συνδυασμός από bits
  - δηλαδή ένας δυαδικός αριθμός
- υπάρχουν πολλές αναπαραστάσεις κειμένου
  - οι πιο γνωστές: ASCII, ISO & UNICODE

Αναπαράσταση Κειμένου
Κωδικοποίηση κατά ASCII

American standard Code for Information Interchange, ASCII

- το πρώτο κοινό πρότυπο αναπαράστασης χαρακτήρων κειμένου που χρησιμοποιήθηκε από όλους τους κατασκευαστές υπολογιστών
  - χρησιμοποιήθηκε για πρώτη φορά το 1963 και ευρέως από το 1968 και μετά
- έχει επικρατήσει σαν πρότυπο κωδικοποίησης αρχείων κειμένου

- χρησιμοποιεί **7** bits για την κωδικοποίηση και ένα όγδοο bit για λόγους έλεγχου στη μετάδοση των bits
  - άρα, υπάρχουν **128** (2^7) διαφορετικοί συνδυασμοί των 7 bits
  - οι υπόλοιποι **128** (2^8 =256 συνολικά) συνδυασμοί χρησιμοποιήθηκαν αργότερα για ειδικούς γραφικούς χαρακτήρες ή για χαρακτήρες άλλων αλφάβητων
    - από τον ISO (International Standardization Organization - Διεθνής Οργανισμός Τυποποίησης)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Αναπαράσταση Κειμένου
Κωδικοποίηση κατά ISO και UNICODE

- η κωδικοποίηση ASCII
  - αναπαριστά ικανοποιητικά τους λατινικούς χαρακτήρες της αγγλικής γλώσσας
  - αλλά δεν συμπεριλαμβάνει εθνικούς χαρακτήρες άλλων χωρών

- το ISO 8859 είναι μια 8-bit επέκταση του ASCII που χρησιμοποιεί και τα 8 bits
  - για τους ελληνικούς χαρακτήρες (μονοτονικό σύστημα) υπάρχει το ISO 8859-7

- η κωδικοποίηση UNICODE χρησιμοποιεί 8, 16 ή και 32 bits για την αναπαράσταση χαρακτήρων (2^8, 2^{16} και 2^{32} διαφορετικούς χαρακτήρες, αντίστοιχα)
  - το πρότυπο UTF-8 είναι η κυρίαρχη κωδικοποίηση στον Παγκόσμιο Ιστό (Word Wide Web, WWW)

Αναπαράσταση ακεραίων αριθμών

- όλοι οι φυσικοί αριθμοί με πρόσημο και το μηδέν
  - εύρος: -∞ …, -2, -1, 0, 1, 2, … +∞
- δεν υπάρχει υπολογιστικό σύστημα που να αναπαριστά όλους τους ακεραίους
  - κάθε υπολογιστικό σύστημα χρησιμοποιεί συγκεκριμένο και περιορισμένο πλήθος από bits για την αναπαράσταση των αριθμών
  - το πλήθος αυτό καθορίζει και το εύρος των τιμών των ακεραίων

- υπάρχουν διαφορετικές αναπαραστάσεις για:
  - ακεραίους χωρίς πρόσημο *(μη προσημασμένοι: από 0 έως ∞)*
  - ακεραίους με πρόσημο *(προσημασμένοι: από -∞ έως +∞)*

Αναπαράσταση ακεραίων αριθμών

Μη προσημασμένοι

- ο αριθμός (**N**) των bits που χρησιμοποιείται για την αναπαράσταση καθορίζει και τον μεγαλύτερο αριθμό που μπορεί να αναπαρασταθεί
  - **4** bits:
    - ο μεγαλύτερος ακέραιος είναι $1111_2 = 15_{10} = 2^4-1$
    - εύρος: από **0** έως **15**
  - **8** bits:
    - ο μεγαλύτερος ακέραιος είναι $11111111_2 = 255_{10} = 2^8-1$
    - εύρος: από **0** έως ***255***
  - γενικά, για **N** bits το εύρος των τιμών που έχουμε για ακεραίους χωρίς πρόσημο είναι: *από **0** έως **2^N-1***

☞ για να αποθηκευτεί σε υπολογιστή ένας μη προσημασμένος ακέραιος αριθμός απλά μετατρέπεται στο δυαδικό σύστημα

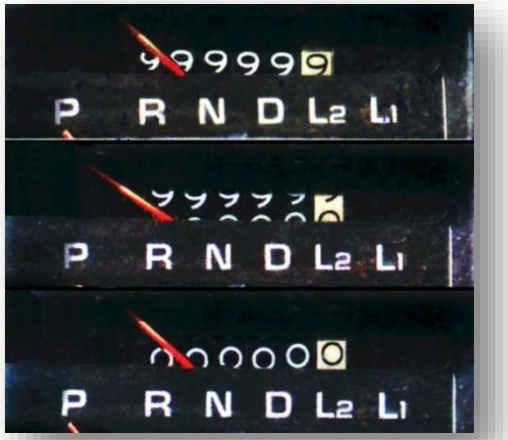
Αναπαράσταση ακεραίων αριθμών

Μη προσημασμένοι - Παραδείγματα

έστω **N** ο αριθμός των bits που χρησιμοποιείται για την αναπαράσταση

- ✓ αν **N** = 8 → ο αριθμός 12_{10} (1100_2) αποθηκεύεται ως: **00001100**
- ✓ αν **N** = 16 → ο αριθμός 12_{10} (1100_2) αποθηκεύεται ως: **0000000000001100**

- ✗ αν **N** = 8 → ο αριθμός 260_{10} (100000100_2) δε μπορεί να αποθηκευτεί
  - ? αν προσπαθήσουμε να τον αποθηκεύσουμε σε 8 bits;
    - 👎 θα αποθηκευτεί **0000100**, δηλαδή ο αριθμός 4_{10} (100_2)
    - ✍ έχουμε το φαινόμενο της **υπερχείλισης** (overflow) → οδηγεί σε **απώλεια** πληροφορίας
- ✓ αν **N** = 16 → ο αριθμός 260_{10} (100000100_2) αποθηκεύεται ως: **0000000100000100**
- ✓ αν **N** = 32 → ο αριθμός 260_{10} (100000100_2) αποθηκεύεται ως: **00000000000000000000000100000100**



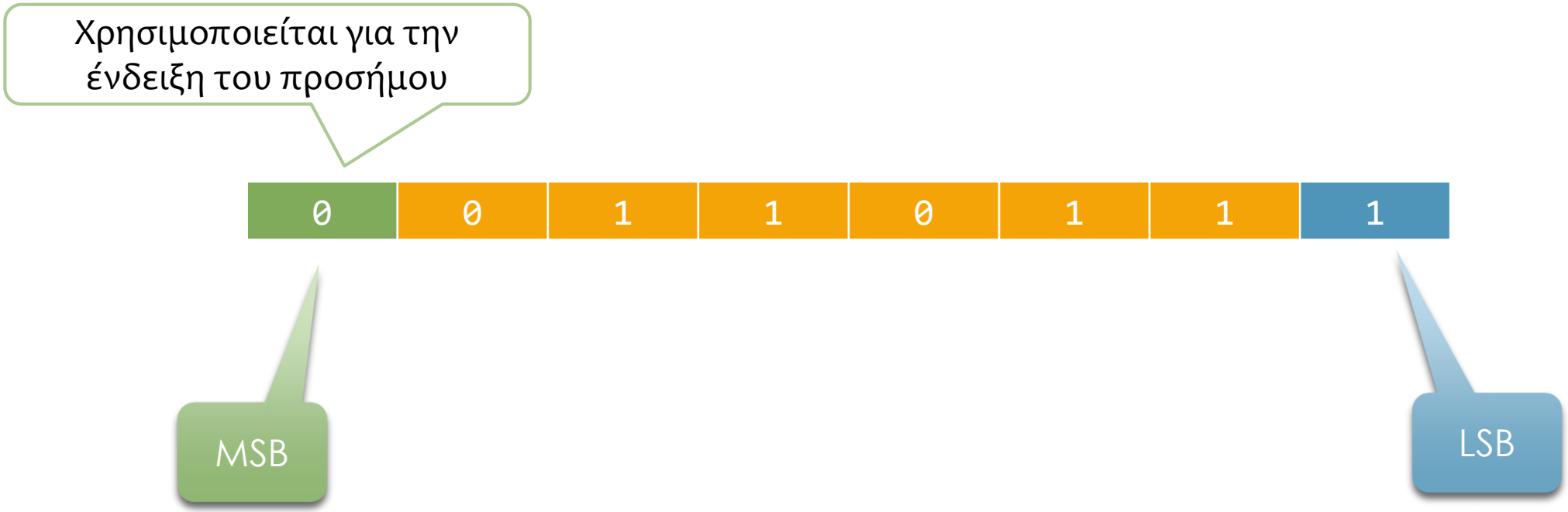
Αναπαράσταση ακεραίων αριθμών
Προσημασμένοι

- αφιερώνεται ένα bit για την ένδειξη του πρόσημου του ακεραίου:
  - **`0`** → θετικός
  - **`1`** → αρνητικός
- τα υπόλοιπα bits χρησιμοποιούνται για την αναπαράσταση του μέτρου του ακεραίου (απόλυτη τιμή)
  - ✍ με ένα bit λιγότερο (αυτό του πρόσημου) το εύρος των αριθμών υποδιπλασιάζεται
- στις κωδικοποιήσεις που θα εξετάσουμε στη συνέχεια αλλάξει μόνο ο τρόπος αναπαράστασης των αρνητικών αριθμών
  (η αναπαράσταση των θετικών ακεραίων είναι η ίδια)

1. **πρόσημο και μέτρο**
2. **συμπλήρωμα ως προς 1**
3. **συμπλήρωμα ως προς 2**

Δυαδική Αναπαράσταση

- δυαδική αναπαράσταση (με κατεύθυνση από αριστερά προς τα δεξιά)
  - το πρώτο ψηφίο είναι το **περισσότερο σημαντικό ψηφίο** (Most Significant Bit – MSB)
  - το τελευταίο ψηφίο είναι το **λιγότερο σημαντικό ψηφίο** (Least Significant Bit - LSB)



Αναπαράσταση ακεραίων αριθμών

Προσημασμένοι - Κωδικοποίηση **πρόσημο και μέτρο**

- το πιο σημαντικό bit (**MSB**) χρησιμοποιείται για την ένδειξη του πρόσημου
- τα εναπομείναντα ψηφία χρησιμοποιούνται για το μέτρο του αριθμού:
  - $+12_{10}$ → **00001100** *(8 bits αναπαράσταση)*
  - -12_{10} → **10001100** *(8 bits αναπαράσταση)*
  - $+260_{10}$ → **0000000100000100** *(16 bits αναπαράσταση)*
  - -260_{10} → **1000000100000100** *(16 bits αναπαράσταση*
- έστω ότι χρησιμοποιούμε κωδικοποίηση πρόσημου-μέτρου με **N** bits
  - ο μέγιστος προσημασμένος ακέραιος είναι:

$$\mathbf{011111\ldots1111}_2 = +(2^{N-1} - 1)_{10}$$

  - ο ελάχιστος προσημασμένος ακέραιος είναι:

$$\mathbf{111111\ldots1111}_2 = -(2^{N-1} - 1)_{10}$$

Αναπαράσταση ακεραίων αριθμών

Προσημασμένοι - Κωδικοποίηση **πρόσημο και μέτρο** II

- έστω ότι χρησιμοποιούμε κωδικοποίηση πρόσημου-μέτρου με **N** bits,
  - το εύρος τιμών που έχουμε είναι: από $-(2^{(N-1)}-1)_{10}$ έως $+(2^{(N-1)}-1)_{10}$

  παραδείγματα:
  - με **8** bits το εύρος τιμών είναι: -2^7-1 έως $+2^7-1$ *(-127 έως +127)*
  - με **16** bits το εύρος τιμών είναι: $-2^{15}-1$ έως $+2^{15}-1$ *(-32767 έως +32767)*

- έχουμε δύο αναπαραστάσεις για το μηδέν:
  - **000…000** ($+0_{10}$) και
  - **100…000** (-0_{10})

Αναπαράσταση ακεραίων αριθμών

Προσημασμένοι - Κωδικοποίηση **συμπληρώματος ως προς 1**

- (πάλι) το πιο σημαντικό bit (**MSB**) χρησιμοποιείται για την ένδειξη του πρόσημου
- για την αναπαράσταση των αρνητικών ακεραίων χρησιμοποιούμε το συμπλήρωμα ως προς 1 της δυαδικής αναπαράστασης του αριθμού
  - το συμπλήρωμα ως προς 1 ενός δυαδικού αριθμού βρίσκεται εύκολα αν αλλάξουμε
    - όλα τα ψηφία 1 σε 0 και
    - όλα τα ψηφία 0 σε 1

παραδείγματα:

- $+12_{10}$ (1100_2) → **00001100** *(8 bits αναπαράσταση)*
  - όμοια με την κωδικοποίηση προσήμου και μέτρου
- -12_{10} θα παρασταθεί ως: **11110011**
  - καθώς το συμπλήρωμα ως προς 1 του (0001100_2) είναι το (1110011_2)

Αναπαράσταση ακεραίων αριθμών

Προσημασμένοι - Κωδικοποίηση **συμπληρώματος ως προς 1** II

- έστω ότι χρησιμοποιούμε κωδικοποίηση συμπληρώματος ως προς 1 με **N** bits
  - μέγιστος προσημασμένος ακέραιος: **0**1111…1111 $= +(2^{(N-1)} - 1)_{10}$
  - ελάχιστος προσημασμένος ακέραιος: **1**0000…0000 $= -(2^{(N-1)} - 1)_{10}$
    - όπου το **0000…0000** είναι συμπλήρωμα ως προς 1 του **1111…1111**
  - το εύρος τιμών που έχουμε είναι: από $-(2^{(N-1)}-1)_{10}$ έως $+(2^{(N-1)}-1)_{10}$

- (και πάλι) έχουμε δύο αναπαραστάσεις για το μηδέν:
  - **0**00000…00000 $(+0_{10})$ και
  - **11111…11111** (-0_{10})
    - όπου το **1111…11111** είναι συμπλήρωμα ως προς 1 του **0000…00000**

Αναπαράσταση ακεραίων αριθμών

Προσημασμένοι - Κωδικοποίηση **συμπληρώματος ως προς 2**

- (πάλι) το πιο σημαντικό bit (**MSB**) χρησιμοποιείται για την ένδειξη του πρόσημου
- για την αναπαράσταση των αρνητικών ακεραίων χρησιμοποιούμε το συμπλήρωμα ως προς 2 της δυαδικής αναπαράστασης του αριθμού
  - το συμπλήρωμα ως προς 2 ενός δυαδικού αριθμού βρίσκεται μετατρέποντας
    - όλα τα ψηφία 1 σε 0 και
    - όλα τα ψηφία 0 σε 1, και
    - στο αριθμό που προκύψει προσθέσουμε τον αριθμό 1_2

το συμπλήρωμα ως προς 1 του αριθμού

παραδείγματα:

- $+12_{10}$ (1100_2) → **00001100** *(8 bits αναπαράσταση)*
  - όμοια με την κωδικοποίηση συμπληρώματος ως προς 1
- -12_{10} θα παρασταθεί ως: **11110100**
  - καθώς το συμπλήρωμα ως προς 1 του (0001100_2) είναι το (1110011_2) και
  - (1110011_2) + (1_2) = 1110100_2

Αναπαράσταση ακεραίων αριθμών

Προσημασμένοι - Κωδικοποίηση **συμπληρώματος ως προς 2** II

- έστω ότι χρησιμοποιούμε κωδικοποίηση συμπληρώματος ως προς 2 με **N** bits
  - μέγιστος προσημασμένος ακέραιος: **0**1111…1111 $= +(2^{(N-1)} - 1)_{10}$
  - ελάχιστος προσημασμένος ακέραιος: **1**0000…0001 $= -(2^{(N-1)} - 1)_{10}$
    - όπου το **10000…0001** είναι συμπλήρωμα ως προς 2 του **11111…1111**
  - το εύρος τιμών που έχουμε είναι: από $-(2^{(N-1)})_{10}$ έως $+(2^{(N-1)}-1)_{10}$

✍ έχουμε μία αναπαράσταση για το μηδέν: **000000…00000** $(+0_{10})$

✍ πρόκειται για τον τρόπο αναπαράστασης ακεραίων με πρόσημο

Αναπαράσταση ακεραίων αριθμών

Κωδικοποίηση - Παράδειγμα με 4 bits

	μέτρου και προσήμου	συμπληρώματος ως προς 1	συμπληρώματος ως προς 2
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000, 1000	0000, 1111	0000
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	-	-	1000

Αναπαράσταση ακεραίων αριθμών

Κωδικοποίηση - Παράδειγμα με 8 bits

	μέτρου και προσήμου	συμπληρώματος ως προς 1	συμπληρώματος ως προς 2
127	01111111	01111111	01111111
126	01111110	01111110	01111110
...	...	...	...
2	00000010	00000010	00000010
1	00000001	00000001	00000001
0	00000000, 10000000	00000000, 11111111	00000000
-1	10000001	11111110	11111111
-2	10000010	11111101	11111110
...	...	...	...
-126	11111110	10000001	10000010
-127	11111111	10000000	10000001
-128	-	-	10000000

Δεδομένα, Τύποι και Τιμές

Δεδομένα (data)

- απαραίτητα στοιχεία ενός προγράμματος
- βασικές λειτουργίες ενός προγράμματος:
  1. αποθήκευση δεδομένων
     - απαιτείται δέσμευση χώρων στη μνήμη
  2. επεξεργασία δεδομένων και
  3. εξαγωγή αποτελεσμάτων (δηλαδή άλλα δεδομένα)
- τύποι δεδομένων
  - οι γλώσσες προγραμματισμού μας δίνουν τη δυνατότητα να δηλώσουμε κάποιους τύπους χώρων μνήμης
    - ✍ μπορούμε να φτιάξουμε και δικούς μας!
    - ονομάζονται ανάλογα με τη χρήση τους:
      - σταθερές
      - μεταβλητές

Τύποι Δεδομένων

- η αναπαράσταση δεδομένων στον υπολογιστή διαφέρει, ανάλογα με την επεξεργασία που θέλουμε να κάνουμε

- το μέγεθος της μνήμης που δεσμεύεται για κάποιο τύπο είναι περιορισμένο
  - άρα, οι τιμές στα δεδομένα που αναπαριστούνται σε κάθε τύπο είναι περιορισμένες

- το εύρος των διαφόρων τύπων δεδομένων μπορεί να είναι διαφορετικό
  - εξαρτάται από το μεταγλωττιστή και τον υπολογιστή

Τύποι Δεδομένων

Βασικοί & Σύνθετοι

βασικοί τύποι δεδομένων

1. χαρακτήρες (`char`)
2. ακέραιοι (`int`)
3. κινητής υποδιαστολής (`float`)

✍ υπάρχουν ακόμα:

- χαρακτηρισμός προσήμου: `singed` / `unsigned` (χαρακτήρες ή ακέραιοι)
- προσδιορισμός μεγέθους: `short`, `long` (ακέραιοι), `double` (κινήτης υποδιαστολής)

σύνθετοι τύποι δεδομένων:

- πίνακες, δείκτες, ενώσεις, δομές

Τύποι Δεδομένων

Βασικοί

ο τύπος μιας μεταβλητής καθορίζει το είδος των τιμών που μπορούν να αποθηκευτούν σε αυτή

- η C παρέχει ένα σύνολο τύπων
  - ονομάζονται ενσωματωμένοι (built-in)
- οι προγραμματιστές μπορούν να ορίσουν νέους τύπους
  - ονομάζονται user-defined

ενσωματωμένοι τύποι της C

τύπος	μέγεθος (συνήθως)	περιγραφή	σταθερά
`char`	1 byte	χαρακτήρας	`'a', 'x', '4', '\n', '$'`
`int`	4 bytes	ακέραιος	`0, 1, 123, -6, 034, 0xa3`
`short int`	2 bytes		
`long int`	4 bytes		
`float`	4 bytes	κινητής υποδιαστολής	`1.2, 13.345, .3, -0.54, 1.2e3, .3F`
`double`	8 bytes		

Χώροι μνήμης
Μεταβλητές - Δήλωση

- μία μεταβλητή είναι κάποια μνήμη που διατηρεί την τιμή ενός συγκεκριμένου τύπου
- η δήλωση των μεταβλητών γίνεται συνήθως στην αρχή του προγράμματος
  - ☞ κανόνας: πριν χρησιμοποιηθεί πρέπει να έχει δηλωθεί!
  - σύνταξη:

```
τύπος όνομα_μεταβλητής;
```

  - παραδείγματα:

```
int alpha;              // δήλωση ενός ακεραίου αριθμού
float delta = 5.6;      // δήλωση και αρχικοποίηση ενός αριθμού κινητής υποδιαστολής
char flag;              // δήλωση ενός χαρακτήρα
```

- κάθε μεταβλητή χαρακτηρίζεται από το όνομα της, τον τύπο δεδομένων και τη διεύθυνση της

Χώροι μνήμης
Μεταβλητές - Δέσμευση χώρου

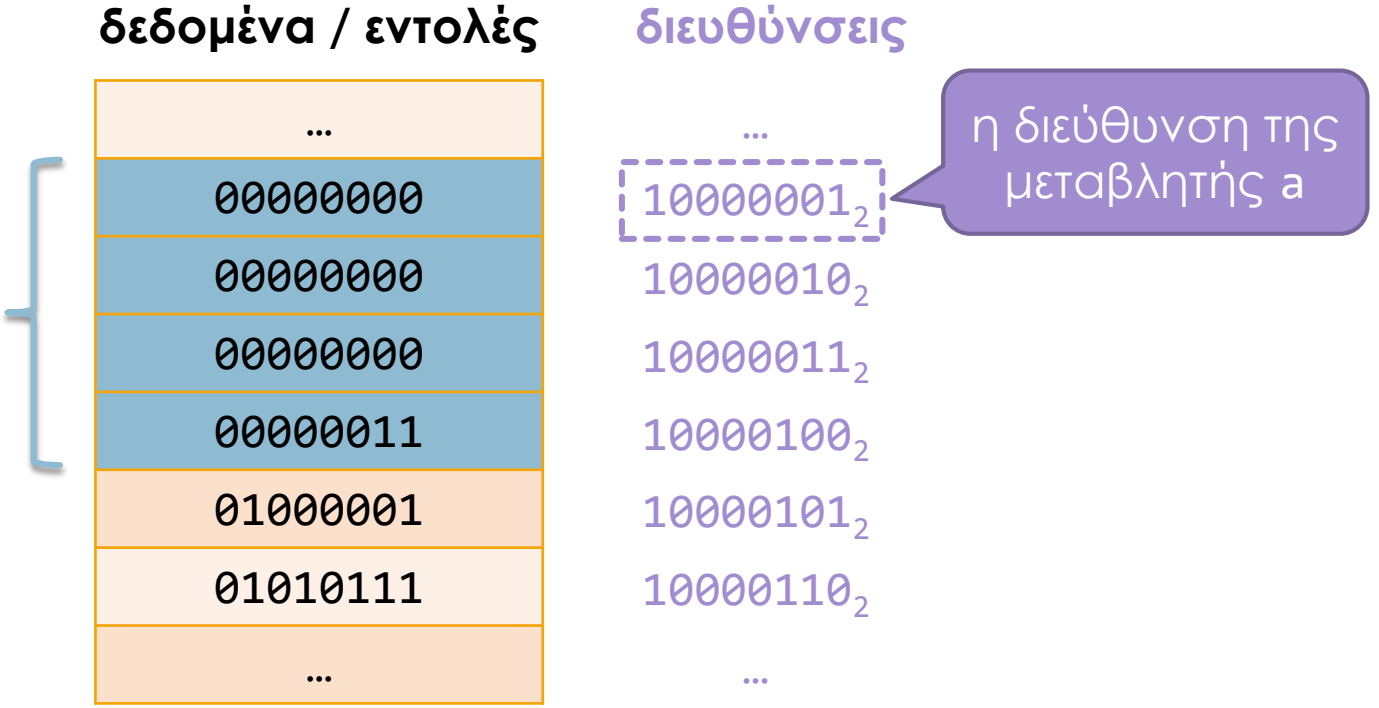
η δήλωση μιας μεταβλητής αποτελεί:

1. τη δέσμευση χώρου μνήμης για τη μεταβλητή
2. την απόδοση του ονόματος της μεταβλητής σε αυτό το χώρο

παράδειγμα:

```
int a = 3;
```

ο χώρος μνήμης της μεταβλητής a



Μεταβλητές
Ανάθεση τιμών

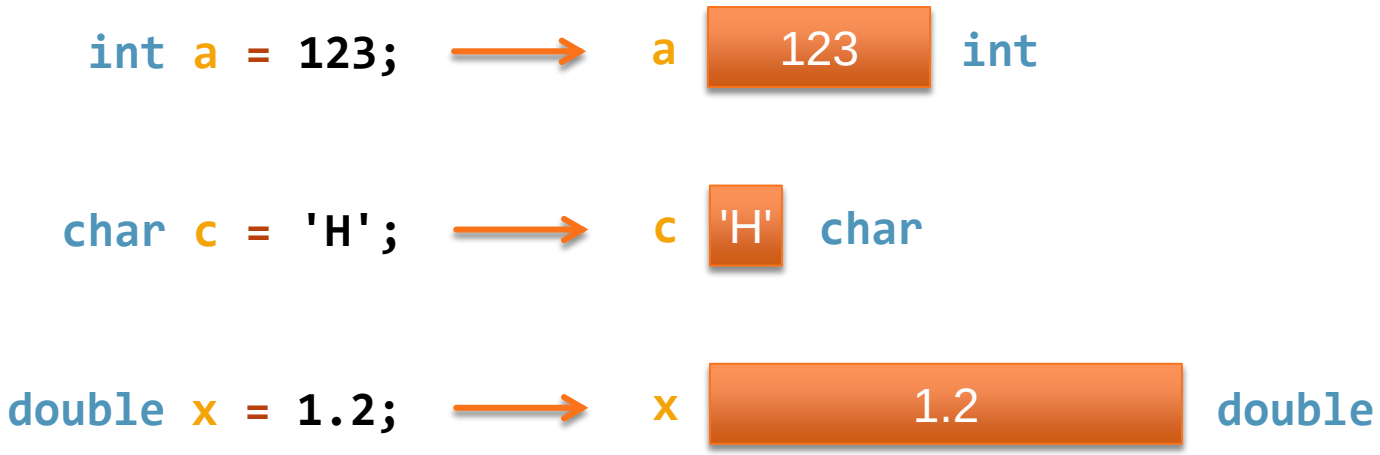
- χρησιμοποιούμε το σύμβολο ίσον ('=')
  - η χρήση του συμβόλου '=' έχει διαφορετική σημασία από ότι στα μαθηματικά
  - το σύμβολο '=' ονομάζεται τελεστής ανάθεσης ή εκχώρησης

- παράδειγμα:

```
a = 52;                 // ανάθεση της τιμής 52 στη μεταβλητή a

a = a + 1;              /* "αυξάνω" κατά μία μονάδα την τιμή της μεταβλητής a →
                           δηλαδή η νέα τιμή της a μετά την εκτέλεση της
                           συγκεκριμένης εντολής θα είναι 53! */
```

Μεταβλητές
Δήλωση και αρχικοποίηση



Μεταβλητές
Αρχικοποίηση και ανάθεση

```
1. int a = 1;
```


```
// το a ξεκινάει με την τιμή 1
```

```
2. a = 10;
```


```
// το a παίρνει την τιμή 10
```

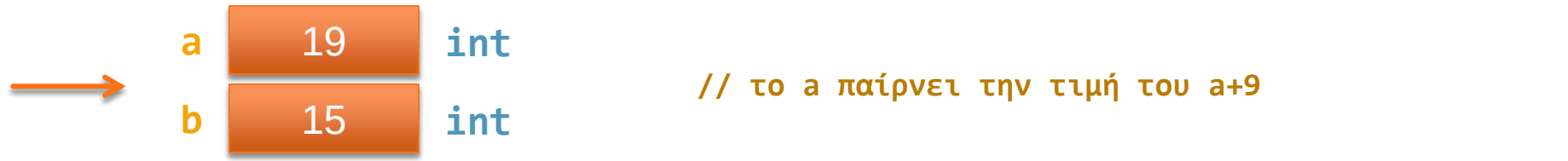
```
3. int b = a;
```


```
// το b ξεκινάει με ένα αντίγραφο της τιμής του a
```

```
4. b = a + 5;
```


```
// το b παίρνει την τιμή του a+5
```

```
5. a = a + 9;
```



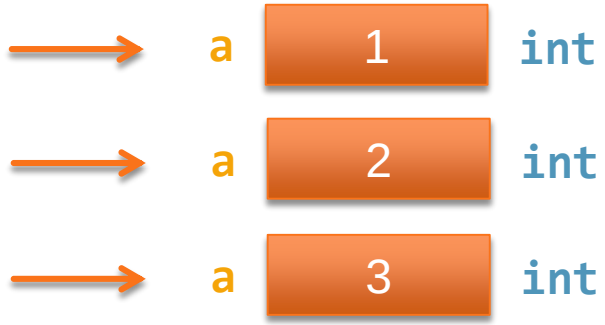

```
// το a παίρνει την τιμή του a+9
```

Μεταβλητές

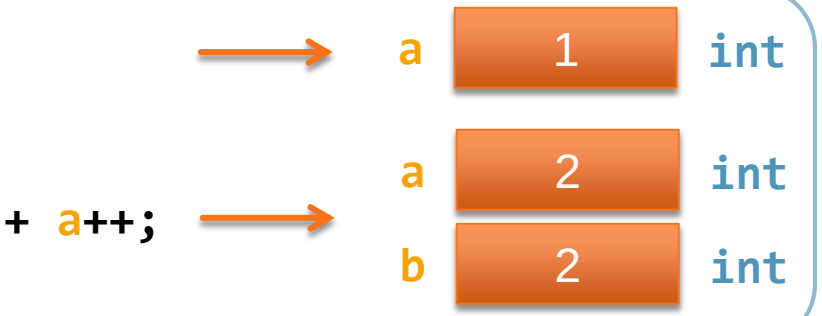
Σύνθετοι τελεστές εκχώρησης ++ και --

Αύξηση /μείωση της μεταβλητής κατά 1 μονάδα. Μπαίνουν είτε **μετά**, είτε **πριν** τη μεταβλητή

```
1. int a = 1;
2. a++;
3. ++a;
```

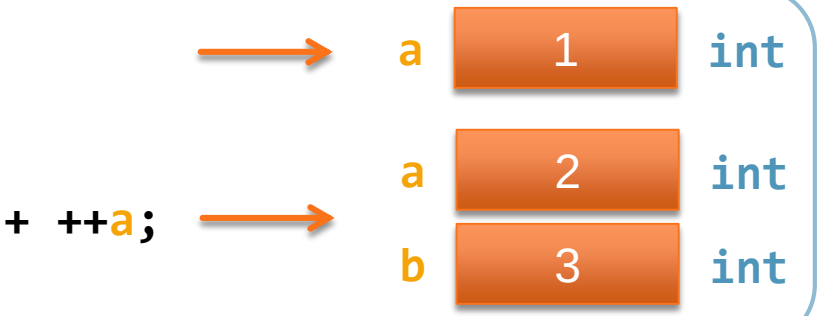



```
1. int a = 1;
2. int b = 1 + a++;
```




```
1. int a = 9;
2. a--;
3. --a;
```


```
1. int a = 1;
2. int b = 1 + ++a;
```



a++ : πρώτα ***χρησιμοποιείται στην παράσταση*** και μετά αυξάνεται κατά 1 μονάδα

++a : πρώτα ***αυξάνεται κατά 1 μονάδα*** και μετά χρησιμοποιείται στην παράσταση

Μεταβλητές

Σύνθετοι τελεστές εκχώρησης += -= *= /= %=

Γενικά, για ένα δυαδικό τελεστή ❖ ισχύει: `a ❖= b` ⇨ `a = a ❖ b`

Παραδείγματα

Έστω ότι `a = 11` και `b = 4`

Αν χρησιμοποιηθεί η εντολή εκχώρησης `a = b`, τότε το `a` θα πάρει την τιμή `4`

Αν χρησιμοποιηθεί η εντολή εκχώρησης `a += b`, τότε το `a` θα πάρει την τιμή `15`

Αν χρησιμοποιηθεί η εντολή εκχώρησης `a -= b`, τότε το `a` θα πάρει την τιμή `7`

Αν χρησιμοποιηθεί η εντολή εκχώρησης `a *= b`, τότε το `a` θα πάρει την τιμή `44`

Αν χρησιμοποιηθεί η εντολή εκχώρησης `a /= b`, τότε το `a` θα πάρει την τιμή `2`

Αν χρησιμοποιηθεί η εντολή εκχώρησης `a %= b`, τότε το `a` θα πάρει την τιμή `3`

Μεταβλητές
Ανάθεση τιμών II

- η γενική μορφή της εντολής εκχώρησης είναι:

```
όνομα_μεταβλητής = έκφραση;
```

  - όπου:
    1. αποτιμάται το δεξί μέλος της ανάθεσης (**έκφραση**) σε μία τιμή και
    2. η τιμή αυτή ανατίθεται στη μεταβλητή
  - η **έκφραση** μπορεί να είναι
    - μια σταθερή τιμή,
    - μια άλλη μεταβλητή,
    - ένας μαθηματικός τύπος,
    - το αποτέλεσμα κλήσης μιας συνάρτησης
  - ✍ η **έκφραση** θα πρέπει να παράγει ένα αποτέλεσμα που να είναι συμβατό με τον τύπο της μεταβλητής

Μεταβλητές
Πολλαπλές αναθέσεις

- μπορούμε να κάνουμε πολλαπλές αναθέσεις, π.χ.

```
x = y = z = 74;
```

  - οι αναθέσεις γίνονται από δεξιά προς τα αριστερά
    1. δηλαδή πρώτα παίρνει την τιμή **74** η μεταβλητή **z**,
    2. μετά η μεταβλητή **y** και
    3. τέλος η **x**

- είναι σωστή η παρακάτω γραμμή;

```
x = y = 74 = z;
```

  - όχι!
  - θα προκύψει **σφάλμα** κατά τη μεταγλώττιση, καθώς δίνεται εντολή ανάθεση τιμής στο **74**!
    - το **74** **δε** μπορεί να είναι **όνομα** μεταβλητής!

Ονόματα
(για μεταβλητές, συναρτήσεις, τύπους κλπ.)

- ένα όνομα ξεκινά με γράμμα και μπορεί να περιέχει:
  - γράμματα
    - τα πεζά και κεφαλαία γράμματα παίζουν ρόλο (είναι διαφορετικά!)
  - ψηφία
  - χαρακτήρες underscore (κάτω παύλα)
- δεν επιτρέπονται άλλοι ειδικοί χαρακτήρες όπως: #, &, *, +, -, %, οι οποίοι χρησιμοποιούνται για άλλο σκοπό
- μη ξεκινάτε ονόματα με underscore
  - π.χ. `_foo`
  - δεσμεύονται για οντότητες υλοποίησης και συστήματος
- δε μπορούν να χρησιμοποιηθούν ονόματα δεσμευμένες λέξεις της C
  - π.χ. `int`, `if`, `while`, `double`

έγκυρα
`x`
`num_of_elements`
`Fourier_trasnform`
`z2`

μη έγκυρα
`2x`
`time$to$market`
`Start menu`
`hello_!`

Δεσμευμένες λέξεις (keywords)
της C

DATA TYPES	STORAGE CLASSES	STATEMENTS
char	auto	break
double	extern	case
enum	register	continue
float	static	default
int		do
long		else
short		for
struct		goto
union		if
unsigned		return
void		switch
sizeof		while
typedef		

Ονόματα
Συμβουλές

- επιλέξτε ονόματα που σημαίνουν κάτι
  - θα βοηθήσουν τους άλλους να κατανοήσουν το πρόγραμμά σας
  - θα μπορείτε και εσείς μετά από μήνες να καταλάβετε τι γράφετε

επιθυμητά
`partial_sum`
`element_count`
`free_slots`

- οι συντομογραφίες και τα ακρωνύμια μπερδεύουν

- τα σύντομα ονόματα έχουν σημασία μόνο όταν χρησιμοποιούνται με συμβατικό τρόπο
  - `x` → τοπική μεταβλητή
  - `i` → μετρητής βρόχου

μη επιθυμητά		
`ps`	`titps`	`this_is_the_partial_sum`
`ec`	`tnoe`	`the_number_of_elements`
`fs`	`rfsitst`	`remaining_free_slots_in_the_symbol_table`

- μη χρησιμοποιείται υπερβολικά μεγάλα ονόματα

Ονόματα
Συμβουλές - Στυλ

χρησιμοποιούμε:

- underscore για το διαχωρισμό των λέξεων
- χρησιμοποιούμε ένα αρχικό κεφαλαίο γράμμα για τύπους που ορίζουμε εμείς

επιθυμητά	
`partial_sum`	
`Square`	`Graph`

δε χρησιμοποιούμε:

- ονόματα μόνο με κεφαλαία γράμματα
  - διότι θυμίζουν σε μακροεντολές

μη επιθυμητά		
`PARTIAL_SUM`		
`SQUARE`		`GRAPH`
`Name`	`names`	`nameS`
`foo`	`f00`	`fi`
`f1`	`fl`	`fI`

αποφεύγουμε:

- ονόματα που μπορούν να γραφούν και να διαβαστούν με λάθος τρόπο

χαρακτήρες ιδιαίτερα ευάλωτοι σε σφάλματα						
`O`	`o`	`0`	`1`	`l`	`I`	`i`

Τύποι δεδομένων
char

χρησιμοποιείται για να αποθηκεύσει:

- εκτυπώσιμους χαρακτήρες: `A-Z`, `a-z`, `0-9`, `!`, `@`, `#`, `$`, `%`, `^`, `&`, `*`, `(`, `)`, `_`, `-`, `+`, `/`
- και χαρακτήρες ελέγχου (μη εκτυπώσιμους)
  - π.χ. αλλαγή γραμμής, <enter>, <backspace>

- μέγεθος: 8 bits (1 byte)
- εύρος:
  - από -128 έως 127 (με πρόσημο) ή
  - από 0 έως 255 (χωρίς πρόσημο)

} ? γιατί;

✍ η C χειρίζεται τις μεταβλητές τύπου **char** σαν αριθμούς
  - μπορείτε να κάνετε πράξεις με αυτές

Τύποι δεδομένων
char - δήλωση, αρχικοποίηση, χρήση

- παραδείγματα δήλωσης:

```
char ch1;                    // δήλωση
unsigned char ch2;           // δήλωση
char ch3 = 'A';              // δήλωση και αρχικοποίηση
```

- παραδείγματα χρήσης:

```
ch = 'D';                    /* ανάθεση του χαρακτήρα 'D' στη
                                μεταβλητή ch */
printf("the character is: %c", ch);   /* εκτύπωνει στην οθόνη του τερματικού:
                                         the character is D */
```

δηλώνει την εκτύπωση χαρακτήρα

Τύποι δεδομένων
(unsigned/signed) int, short, long

- χρησιμοποιείται για το χειρισμό ακεραίων αριθμών:
  1. με πρόσημο (..., -2, -1, 0, 1, 2, 3, ...) ή
  2. χωρίς πρόσημο (0, 1, 2, 3,...)

τύπος	μέγεθος (συνήθως)	περιγραφή	εύρος
`signed short int`	2 bytes	ακέραιος	-32.768 έως 32.767 (2^{15}-1)
`unsigned short int`			0 έως 65.535 (2^{16}-1)
`singed long int`	4 bytes		-2.147.483.648 έως 2.147.438.647 (2^{31}-1)
`unsigned long int`			0 έως 4.294.967.295 (2^{32}-1)

- εύρος **`int`**: συνήθως ταυτίζεται με τον **`short`** ή τον **`long`**
  - εξαρτάται από το μεταγλωττιστή
  - ✍ γενικά: **`short`** ≤ **`int`** ≤ **`long`**

Τύποι δεδομένων
(unsigned/signed) int, short, long - δήλωση, αρχικοποίηση, χρήση

- παραδείγματα δήλωσης:

```
int a;                          // δήλωση
short counter = 0;              // δήλωση και αρχικοποίηση
long b, c;                      // δήλωση πολλαπλών μεταβλητών σε μία γραμμή
unsigned int num_students;      // δήλωση
```

ισοδύναμα:
long b;
long c;

- παραδείγματα χρήσης:

```
a = 5;                                  // ανάθεση του αριθμού 5 στη μεταβλητή a
printf("the integer is: %d", a);        /* εκτύπωνει στην οθόνη του τερματικού:
                                           the integer is 5 */
```

δηλώνει την εκτύπωση ακεραίου αριθμού

Τύποι δεδομένων
float, double

- χρησιμοποιείται για το χειρισμό πραγματικών αριθμών
  - περιλαμβάνουν υποδιαστολή
  - π.χ. `34.5` ή `23.0` ή `0.987`

το E ή e αναπαριστά το 10

ο αριθμός που ακολουθεί είναι η θετική ή αρνητική δύναμη του 10

τύπος	μέγεθος (συνήθως)	περιγραφή	εύρος
`float`	`4 bytes`	κινητής υποδιαστολής	1.2e-38 έως 3.4e+38
`double`	`8 bytes`		2.3e-308 έως 1.7e+308

- χρησιμοποιούμε τον τύπο
  - `float`: μόνο όταν η ακρίβεια των δεκαδικών ψηφίων δεν είναι τόσο σημαντική
  - `double`: όταν χρειάζόμαστε υψηλή ακρίβεια δεκαδικών ψηφίων

Τύποι δεδομένων

float, double - δήλωση, αρχικοποίηση, χρήση

Προσοχή:
χρησιμοποιήστε την τελεία (.)
και όχι το κόμμα (,)

- παραδείγματα δήλωσης:

ισοδύναμα:
float tasi;
tasi = 5.3;

```
float average;                  // δήλωση
float tasi = 5.3;               // δήλωση και αρχικοποίηση
double error_value;             // δήλωση
double metrisi = 1.2052e+02;    // δήλωση και αρχικοποίηση
```

- παραδείγματα χρήσης:

```
average = 5.0;                  /* ανάθεση του αριθμού 5.0 στη
                                   μεταβλητή average */
printf("the float is: %f", average);    /* εκτύπωνει στην οθόνη του τερματικού:
                                   the float is 5.0 */
```

δηλώνει την εκτύπωση αριθμού κινητής υποδιαστολής

Μεταβλητές
Ανάθεση τιμών - Παρατηρήσεις

- αν μπροστά από μία ακέραια τιμή υπάρχει το ψηφίο **0**, τότε αυτή η τιμή εκλαμβάνεται σαν οκταδικός αριθμός:

```
int a = 0100;                // το alpha είναι 64 και όχι 100
```

- αν μπροστά από μία ακέραια τιμή υπάρχει το **0x** ή το **0X**, τότε αυτή η τιμή ερμηνεύεται σαν δεκαεξαδικός αριθμός:

```
int b = 0x10;                // το beta είναι 16 και όχι 10
```

- η τιμή που δίνεται σε μία μεταβλητή πρέπει να συμβαδίζει με τον τύπο της μεταβλητής:

```
int a = 12.3;                // το alpha γίνεται 12 (Έμμεση μετατροπή τύπου)
```

- η τιμή που εκχωρείται σε μία μεταβλητή πρέπει να είναι μέσα στο επιτρεπτό εύρος τιμών:

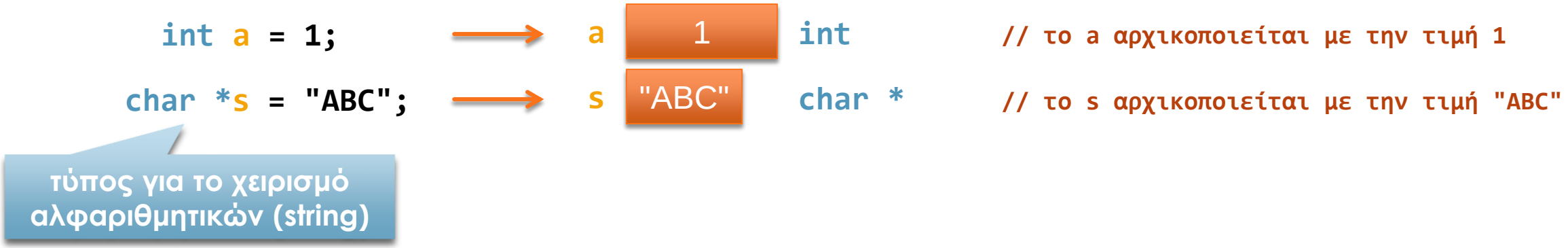
```
unsigned char ch = 340;   // δε γίνεται σωστή ανάθεση
```

- μπορούμε να δώσουμε ακέραια τιμή σε μία float μεταβλητή:

```
float average = 50;      // συνήθως ισοδύναμο με το: float average = 50.0;
                                                    (Έμμεση μετατροπή τύπου)
```

Μεταβλητές
Ανάθεση λάθος τιμών

- μία μεταβλητή διατηρεί τις τιμές ενός συγκεκριμένου τύπου



- δε μπορούμε να τοποθετήσουμε τιμές με το λάθος τύπο σε μία μεταβλητή

```
int a = "ABC";      // σφάλμα: το "ABC" δεν είναι ακέραιος
char *s = 1;        // σφάλμα: το 1 δεν είναι αλφαριθμητικό
```

Μεταβλητές
Μετατροπή τύπου δεδομένων

- όταν οι μεταβλητές που εμπλέκονται σε μία έκφραση είναι του ιδίου τύπου το αποτέλεσμα είναι του ίδιου τύπου

- όμως, όταν
  - κάνουμε πράξεις όπου εμπλέκονται διαφορετικού τύπου μεταβλητές ή
  - καταχωρούμε αποτελέσματα σε διαφορετικού τύπου μεταβλητές,

  πρέπει να είμαστε προσεκτικοί!

Μεταβλητές
Έμμεση μετατροπή τύπου

συχνά σε μία έκφραση εμπλέκονται διαφορετικού τύπου μεταβλητές ή τιμές

- τότε, ο κανόνας είναι: να μετατρέπεται ο τύπος με το μικρότερο μέγεθος στον τύπο με το μεγαλύτερο μέγεθος
  - ☞ ώστε να μη χάνεται πληροφορία
  - ▶ γενικά ισχύει:

```
char < int < long < float < double
```

- ▶ παράδειγμα:

```
int a = 10;          // η μεταβλητή a είναι ακέραιος
float d = 0.5;       // η μεταβλητή d είναι κινητής υποδιαστολής
d = a + d;           // το αποτέλεσμα της έκφρασης είναι κινητής υποδιαστολής (10.5)
```

- εκφράσεις που αναθέτουν έναν τύπο μεγαλύτερου μεγέθους σε μικρότερο (π.χ. `a = d;`)
  - ▶ συνήθως δημιουργούν μια προειδοποίηση (warning) από τον μεταγλωττιστή (compiler)
  - ▶ πρέπει να αποφεύγονται

Μεταβλητές

Πίνακες έμμεσης μετατροπής τύπων

Τύπος	char	int	long	float	double	long double
char	char	int	long	float	double	long double
int	int	int	long	float	double	long double
long	long	long	long	float	double	long double
float	float	float	float	float	double	long double
double	double	double	double	double	double	long double
long double	long double	long double	long double	long double	long double	long double

Μεταβλητές
χώρος στη μνήμη - Μετατροπή τύπου δεδομένων

τύπος	τυπικό μέγεθος	εύρος τιμών
`char`	1 byte	[-127, 127]
`int`	4 bytes	[-2147483648, 2147483647]
`float`	4 bytes	[1.2e-38, 3.4e38]
`double`	8 bytes	[2.3e-308, 1.7e308]
`unsigned char`	1 byte	[0, 255]
`unsigned int`	4 bytes	[0, 4294967295]

ασφαλείς μετατροπές

επισφαλείς μετατροπές

? *Γιατί το εύρος τιμών του float είναι μεγαλύτερο από το εύρος τιμών του ακεραίου;*

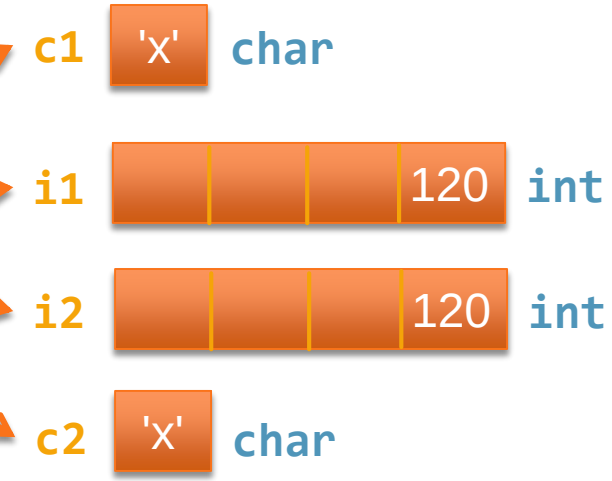
- παρατηρήστε ότι και οι δύο τύποι έχουν το ίδιο τυπικό μέγεθος

Ασφαλείς Μετατροπές

1ο Παράδειγμα

- ασφαλής μετατροπή: δε χάνεται πληροφορία

```
1.  // 1ο παράδειγμα ασφαλών μετατροπών
2.
3.  #include <stdio.h>
4.
5.  int main ()
6.  {
7.     char c1 = 'x';          // το c1 αποκτά την τιμή 'x'
8.     int i1 = c1;            // το i1 αποκτά την τιμή 120
9.     int i2 = 'x';           // το i2 αποκτά την τιμή 120
10.    char c2 = i1;           // το c2 αποκτά την τιμή 'x'
11.    printf("%c %d %c", c1, i1, c2);
12. }                          // εκτυπώνεται: x 120 x
```



- ο χαρακτήρας 'x' είναι η τιμή 120_{10} στο σύνολο χαρακτήρων ASCII

Μία τεχνική λεπτομέρεια

- στη μνήμη τα πάντα είναι απλά bits
- ο τύπος είναι αυτός που δίνει νόημα στα bits

bits / δυαδικό	ακέραιος (int)	χαρακτήρας (char)
01111000	120	'x'
01100001	97	'a'
01000001	65	'A'
00110000	48	'0'

```
1.      ...
2.      char c = 'x';
3.      printf ("%c", c);   // εκτύπωσε την τιμή του χαρακτήρα c, η οποία είναι 'x'
4.      int i = c;
5.      printf ("%d", i);   // εκτύπωσε την ακέραια τιμή του χαρακτήρα c, η οποία είναι 120
6.      ...
```

- ☞ αυτό ισχύει και στον «αληθινό κόσμο»!
  - ? τι σημαίνει 42; → δε γνωρίζουμε μέχρι να μάθουμε τη μονάδα μέτρησης που χρησιμοποιείται
    - ? μέτρα; πόδια; μοίρες; βαθμοί κελσίου; το νούμερο μίας οδού;

Ασφαλείς Μετατροπές

2ο Παράδειγμα

- ασφαλής μετατροπή: δε χάνεται πληροφορία

```
1.  // 2ο παράδειγμα ασφαλών μετατροπών
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.     double d1 = 2.3;
8.     double d2 = d1 + 2;              // το 2 μετατρέπεται σε 2.0 πριν την πρόσθεση
9.     if (d1 < 0)                      // το 0 μετατρέπεται σε 0.0 πριν τη σύγκριση
10.       printf("Το d1 είναι αρνητικό");
11. }
```

- παρατήρηση: για ένα πραγματικά μεγάλο ακέραιο (`int`), θα μπορούσε να έχουμε κάποια απώλεια ακρίβειας κατά τη μετατροπή σε `double` → επισφαλής μετατροπή

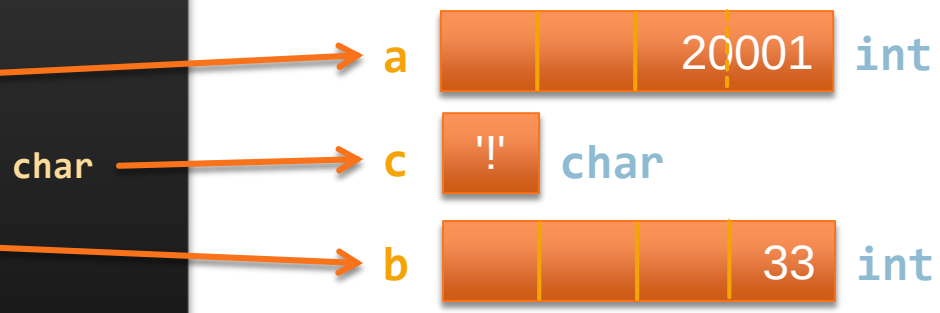
Ασφάλεια Τύπων

Καταστρατήγηση - Επισφαλής μετατροπή ή «Μετατροπές περιορισμού»

- επισφαλής μετατροπή: μία τιμή μπορεί να μετατραπεί (με δυναμικό τρόπο) σε μία τιμή άλλου τύπου που δεν είναι ίση με την αρχική τιμή

bits / δυαδικό	int	char
01001110 00100001	20001	
00100001	33	'!'

```
1.  // Παράδειγμα επισφαλούς μετατροπής
2.
3.  #include <stdio.h>
4.
5.  main ()
6.  {
7.      int a = 20001;
8.      char c = a;          // προσπάθησε να "συμπιέσεις" int σε char
9.      int b = c;
10.     if (a != b)
11.         printf("ωχ!: %d != %d\n", a, b);
12.     else
13.         printf("Τρομερό! Έχουμε μεγάλους χαρακτήρες\n");
14. }
```



Μεταβλητές
Ρητή μετατροπή τύπου

- πολλές φορές κάποιος τύπος δεδομένων πρέπει να μετατραπεί σε κάποιο άλλο τύπο
  - το κάνουμε κυρίως για την αποφυγή απωλειών κλασματικών μερών

- στη C είναι δυνατόν να επιβληθούν τέτοιου είδους μετατροπές
  - αυτό γίνεται βάζοντας τον τύπο δεδομένων μέσα σε παρενθέσεις μπροστά από τη μεταβλητή (που έχει δηλωθεί με διαφορετικό τύπο)
  - παράδειγμα:

```
int a;
(float) a;          // μετατρέπει την τιμή της a σε float
```

Μεταβλητές
Ρητή μετατροπή τύπου - Παράδειγμα

```
1.  // Το πρόγραμμα εκτυπώνει το λόγο δύο ακεραίων στο παράθυρο του τερματικού
2.  #include <stdio.h>
3.
4.  main()
5.  {
6.     int x = 32;
7.     int y = 10;
8.     float division;
9.     division = x / (float) y;
10.    printf ("Result = %f\n", division);
11. }
```

- μετατροπή τύπου (type casting)
  - **(float) y** → μετατρέπει την τιμή της μεταβλητής **y** σε **float**
  - αν δε γίνει η μετατροπή: η διαίρεση μεταξύ ακεραίων μας δίνει πάντα ακέραιο
    - π.χ. το **32/10** δίνει αποτέλεσμα **3** και όχι **3.2**! *(ενώ **32/(float)10** → **3.2**)*

Τύποι δεδομένων

ο τύπος char είναι int

```
1.      ...
2.      char c = 'x';
3.      printf ("%c ", c);    // εκτύπωσε την τιμή του χαρακτήρα c, δηλαδή το 'x'
4.      printf ("%d\n", c);   // εκτύπωσε την ακέραια τιμή του χαρακτήρα c, η οποία είναι 120
5.      ...
```

```
1.      ...
2.      int i = 120;
3.      printf ("%c ", i);       // εκτύπωσε τον χαρακτήρα του ακεραίου i, ο οποίος είναι 'x'
4.      printf ("%d \n ", i);    // εκτύπωσε την (ακέραια) τιμή του i, δηλαδή το 120
5.      ...
```

- το αποτέλεσμα και των δύο προγραμμάτων είναι: **x 120**

Τύποι δεδομένων
ο τελεστής sizeof()

- το μέγεθος των διαφόρων τύπων δεδομένων εξαρτάται από το μεταγλωττιστή
- με τη χρήση του τελεστή **`sizeof()`**
  - ✍ (που μπορεί να μοιάζει με συνάρτηση, αλλά δεν είναι)

μπορούμε να μάθουμε το μέγεθος κάθε μεταβλητής (και σταθεράς) ή τύπου δεδομένων στο μεταγλωττιστή που χρησιμοποιούμε

Τύποι δεδομένων

ο τελεστής sizeof() - μεταβλητής και σταθεράς

- με τη χρήση του τελεστή **sizeof()** μπορούμε να μάθουμε το μέγεθος κάθε μεταβλητής και σταθεράς στο μεταγλωττιστή που χρησιμοποιούμε

```
1.  #include <stdio.h>
2.
3.  main()
4.  {
5.     int x = 32;
6.     double average = 5.0;
7.     printf ("%d\n", sizeof(x+average));
8.  }
```

```
1.  #include <stdio.h>
2.
3.  main()
4.  {
5.     int x = 32;
6.     double average = 5.0;
7.     double n = x + average;
8.     printf ("%d\n", sizeof(n));
9.  }
```

- η **sizeof()** υπολογίζει το μέγεθος κάποιας μεταβλητής ή σταθεράς, αλλά μπορεί να πάρει σαν παραμέτρους και τύπους δεδομένων

Τύποι δεδομένων
ο τελεστής sizeof() - τύπου

- με τη χρήση του τελεστή **sizeof()** μπορούμε να μάθουμε το μέγεθος κάθε τύπου δεδομένων στο μεταγλωττιστή που χρησιμοποιούμε

```
1.  #include <stdio.h>
2.
3.  main()
4.  {
5.     printf("Size of type char: %d bytes\n", sizeof(char));
6.     printf("Size of type int:  %d bytes\n", sizeof(int));
7.     printf("Size of type short: %d bytes\n", sizeof(short));
8.     printf("Size of type long: %d bytes\n", sizeof(long));
9.     printf("Size of type unsigned char: %d bytes\n", sizeof(unsigned char));
10.    printf("Size of type unsigned int: %d bytes\n", sizeof(unsigned int));
11.    printf("Size of type float: %d bytes\n", sizeof(float));
12.    printf("Size of type double: %d bytes\n", sizeof(double));
13. }
```