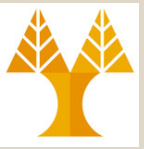
Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου



ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

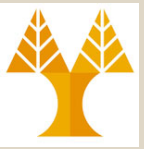
Διάλεξη 3: Εισαγωγή - Τύποι, Πίνακες, Συναρτήσεις

(Κεφάλαια 7-8-9, KNK-2ED)

Δημήτρης Ζεϊναλιπούρ

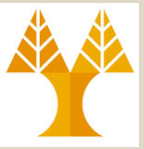
http://www.cs.ucy.ac.cy/courses/EPL232

Εισαγωγική Επισήμανση



- Σε αυτή τη διάλεξη θα έχουμε την ευκαιρία να δούμε τους **βασικούς τύπους** της C σε x86 και x64 μοντέλα**, πίνακες** και **συναρτήσεις.**
- Ευτυχώς και πάλι, **αρκετά σημεία μοιάζουν** με την JAVA, συνεπώς θα εστιαστούμε περισσότερο στα **νέα στοιχεία** (νέους τελεστές και συναρτήσεις), **χωρίς πολλά παραδείγματα**.
- Τις έννοιες αυτές θα έχετε τη δυνατότητα να τις **εμπεδώσετε** μέσω των **εργαστηριακών εργασιών** των πρώτων εβδομάδων και της **Άσκησης 1.**

Περιεχόμενο Διάλεξης 3



- **Βασικοί Τύποι – Basic Types (Κεφ. 7)**
  - int, char, float, double, signed/unsigned, x86/x64
  - Υπερχείλιση τύπου, μετατροπή τύπου (έμμεση, ρητή), ορισμός τύπου, τελεστής sizeof
- **Πίνακες – Arrays (Κεφ. 8)**
  - Δήλωση, Αρχικοποίηση, sizeof,
  - Πολυδιάστατοι & μεταβλητού-μήκους πίνακες (C99),
- **Συναρτήσεις - Functions (Κεφ. 9)**
  - Ορισμός, Κλήση, Πρότυπα, Ορίσματα, Πέρασμα Τιμών Δια Τιμής, Κοινά Λάθη, Πέρασμα Πινάκων δια-διεύθυνσης, static, return, exit,
  - Αναδρομή.

Βασικοί Τύποι Δεδομένων
Τι γνωρίζαμε μέχρι σήμερα;

Σε Αρχιτεκτονική 32 bit (x86) – Γνωστό ως ILP32

Τύπος	Bytes	Εύρος Τιμών
char (χαρακτήρας)	1	-128 … 127
unsigned char	1	0..255
short (ακέραιος)	2	–32,768 to 32,767
int, long [int] (ακέραιος) [..]: προαιρετικό	4,4 (x86)	-2,147,483,648 to 2,147,483,647
long long [int] (ακέραιος)	8	2^{64}
float (πραγματικός)*	4	3.4E+/-38 (7 digits)
double (πραγματικός)	8	1.7E+/-308 (15 digits)

* Εάν αποθηκεύσω πραγματικό με τιμή 0.1 μπορεί αργότερα να βρώ ότι έχει τιμή 0.09999999999999987, λόγω λάθους στρογγυλοποίησης

Βασικοί Τύποι Δεδομένων (Τι Ισχύει Σήμερα)

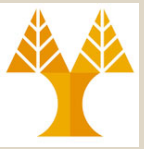
- Εκτός από το διαδεδομένο **ILP32(4,4,4) [IntLongPointer]** υπάρχει και το παλαιότερο:
  - LP32 (2,4,4): π.χ., σε Windows 3.1
- Εκτός από το διαδεδομένο **LP64 (4,8,8)** υπάρχουν και τα ακόλουθα σε περιβάλλον UNIX:
  - **ILP64(8,8,8)**, όπου υπάρχει και το `_int32 (4)`
  - **LLP64(4,4,8)**, όπου υπάρχει και το `long long (8)`

Size in bits

Datatype	ILP32	LP32	LP64	ILP64	LLP64
char	**8**	8	**8**	8	8
short	**16**	16	**16**	16	16
_int32	**-**	-	**-**	32	
int	**32**	16	**32**	64	32
long	**32**	32	**64**	64	32
_int64	**-**	-	**-**	-	64
pointer	**32**	32	**64**	64	64

Μηχανές Lab (Linux, Νέα IntelMacs, κτλ.)

Βασικοί Τύποι Δεδομένων



```
#include <stdio.h>

#if __LP32__ // (2,4,4)
    #define MODEL "LP32 Model"
#elif __ILP32__ // (4,4,4)
    #define MODEL "ILP32 Model"
#elif __LP64__ // (4,8,8)
    #define MODEL "LP64 Model"
#elif __LLP64__ // (4,4,8)
    #define MODEL "LLP64 Model"
#elif __ILP64__ // (8,8,8)
    #define MODEL "ILP64 Model"
#endif

int main() {

    printf("%s", MODEL);
    printf("int:%ld, long:%ld, ptr:%ld, ll:%ld",
       sizeof(int), sizeof(long), sizeof(int *), sizeof(long long));
    return 0;
}
```

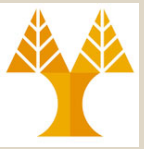
Preprocessor directives, αγνοήστε τα προς το παρόν!

Δοκιμή σε διάφορα μηχανήματα:

```
b103ws1 (Linux 2.6.18)# LP64 Model
int:4, long:8, ptr:8, ll:8

MacOSX (Darwin 10.8)# LP64 Model
int:4, long:8, ptr:8, ll:8
```

Δείκτης: Αγνοήστε προς το παρόν

Τύποι Ακέραιων Αριθμών (Integer Types)



- Η C υποστηρίζει **τύπους ακεραίων (integer types)** και τύπους **πραγματικών (floating types)** δεδομένων.
  - Οι ακέραιοι τύποι με τη σειρά τους χωρίζονται σε **προσημασμένοι (signed)** και **μη-προσημασμένοι (unsigned)**.
- Εξορισμού οι μεταβλητές integer είναι signed στη C,
  - δηλ., το αριστερότερο (ανάλογα με τη διάταξη των bytes) bit αποθηκεύει το πρόσημο.

- Για να πούμε του μεταγλωττιστή ότι μια μεταβλητή δεν έχει πρόσημο, πρέπει να τη δηλώσουμε με **`unsigned`**.
- Οι Unsigned ακέραιοι χρησιμοποιούνται για χαμηλού επιπέδου εφαρμογές και προγραμματισμό συστημάτων.

Τύποι Ακέραιων Αριθμών (Integer Types)

- Οι **οριακές τιμές** ενός τύπου βρίσκονται μέσα στη `limits.h` βιβλιοθήκη
  - Δοκιμάστε σε ένα κέλυφος unix «`man limits.h`»
  - Για παράδειγμα: `INT_MAX, INT_MIN, UINT_MAX, LLONG_MAX, …`
- **Σταθερές:** Δεδομένα τα οποία δεν μεταβάλλονται κατά την εκτέλεση ενός προγράμματος.
  - «`#define TRUE 1`» **ή** «`const int a =1;`»
  - Για να δηλώσετε στον μεταγλωττιστή διαφορετικό τύπο:
    - **Long:** `15L (dec.) 0377L (oct.) 0x7fffL (hex.)`
    - **Unsigned:** `15U 0377U 0x7fffU`
    - **Combined:** `0xffffffffUL`

Τύποι Ακέραιων Αριθμών (Integer Types)

- **Υπερχείλιση Ακεραίου:** Όταν εφαρμόζονται **πράξεις** σε κάποιο αριθμητικό τύπο, ενδέχεται το **αποτέλεσμα** να είναι μεγαλύτερο από τον διαθέσιμο χώρο αναπαράστασης.
  - Π.χ., πρόσθεση 2,147,483,647 + 2,147,483,647
- **Αποτέλεσμα;**

  - ***Signed integers***: Η συμπεριφορά είναι μη-προσδιορισμένη (undefined).
  - **Unsigned integers**, Η συμπεριφορά είναι **προσδιορισμένη (defined) [ίσως και πάλι όχι χρήσιμη]**
    - δηλ., παίρνουμε το **σωστό αποτέλεσμα modulo 2^n,** όπου ***n*** είναι ο **αριθμός των bits** που χρησιμοποιούνται για αναπαράσταση του αποτελέσματος.
    - Σημειώστε ότι τα υπόλοιπα bits ΔΕΝ θα καταλάβουν διπλανά bits στη μνήμη, σε αντίθεση με την υπερχείλιση συμβολοσειρών / πινάκων που θα δούμε αργότερα.

Τύποι Ακέραιων Αριθμών (Integer Types)

- Για να διαβάσετε ή να γράψετε ένα ***short* integer**, τοποθετήσετε το γράμμα **`h`** μπροστά από τα `d`, `o`, `u` (`unsigned`), ή `x`:

```
short s;

scanf("%hd", &s);
printf("%hd", s);
```

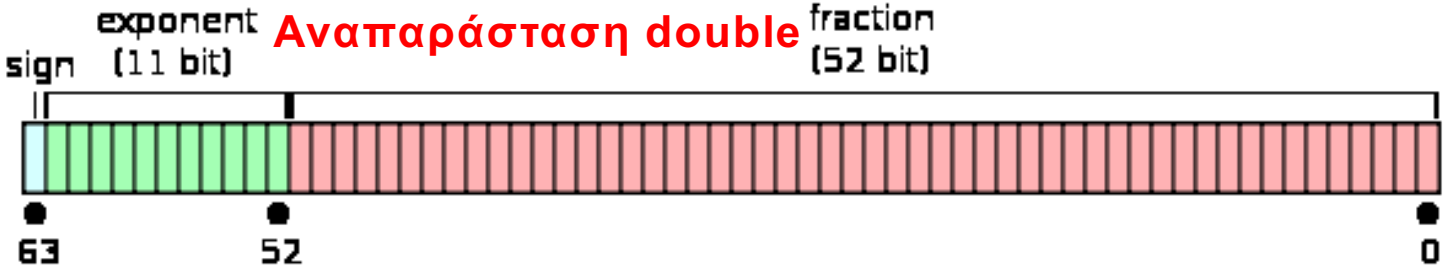
- Για να διαβάσετε ή να γράψετε ένα ***long* integer**, τοποθετήσετε το γράμμα **`l`** (“ell,” όχι “one”) μπροστά από τα `d`, `o`, `u`, or `x`.
- Για να διαβάσετε ή να γράψετε ένα ***long long* integer (C99 only)**, τοποθετήσετε το γράμμα **`ll`** μπροστά από τα `d`, `o`, `u`, or `x`.

int (c, java) vs. Integer (java)

- **int** είναι βασικός τύπος (**primitive type**).
  - Μεταβλητες int αναπαρίστανε στη μνήμη τη δυαδική αναπαράσταση της τιμής (βλέπε προηγούμενες διαφάνειες).
  - Στη JAVA το int.parseInt("1") δεν κάνει νόημα εφόσον το int ΔΕΝ είναι κλάση (class) και συνεπώς δεν έχει μεθόδους μεταποίησης της τιμής του ακεραίου που εκφράζει (π.χ., intValue, toString, equals, byteValue, hashcode)
- **Integer** είναι κλάση (**class**), καθόλου διαφορετικά από τις υπόλοιπες κλάσεις στη JAVA.
  - Μεταβλητές τύπου Integer φυλάνε αναφορές σε αντικείμενα τύπου Integer.
  - π.χ., **Integer.parseInt("1")** είναι κλήση σε στατική μέθοδο “public static int parseInt(String s)”
- Η ίδια συζήτηση ισχύει και για τους υπόλοιπους τύπους που θα δούμε πιο κάτω.

Τύποι Πραγματικών Αριθμών (Floating Types)

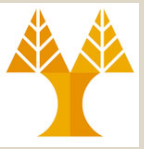
- Μεγέθη (`float.h`):
  - `float (32 bits)` είναι κατάλληλο όταν το μέγεθος ακριβείας (precision) δεν είναι κρίσιμο (π.χ., θερμοκρασία)
  - `double (64 bits)` παρέχει αρκετή ακρίβεια για τα πλείστα προγράμματα (π.χ., money).
  - `long double (80 ή 128 bits)` χρησιμοποιούνται σπάνια.
- Αναπαράσταση: IEEE Standard 754 (aka IEC 60559).



Type	*Smallest Positive Value*	*Largest Value*	*Precision*
`float`	1.17549×10^{-38}	3.40282×10^{38}	6 digits
`double`	2.22507×10^{-308}	1.79769×10^{308}	15 digits

Ακρίβεια πραγματικού αριθμού

Τύποι Χαρακτήρων (Character Types)



- Οι **χαρακτήρες** στη C είναι όπως σε **κάθε άλλη** γλώσσα που είδατε μέχρι στιγμής.
  - Ειδικότερα είναι **8 bit** και επιδέχεται να χρησιμοποιηθούν με αριθμητικές πράξεις ‘A’+’B’
  - Οι αντιστοιχίσεις των ακέραιων με τους χαρακτήρες ορίζονται από τον πίνακα ASCII.
- Signed και Unsigned Characters:
  - **Signed characters** (κοινή χρήση): έχουν τιμή μεταξύ –128 και 127 (οι αρνητικές τιμές είναι άχρηστες).
  - **Unsigned characters**: τιμές μεταξύ 0 and 255
    - βασική μονάδα αποθήκευσης ενός byte, π.χ., δημιουργία τύπου byte: **typedef** unsigned char **byte**;

ASCII Table (7-bit)

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Printable vs Non-Printable?

Extended ASCII (8-bit): **Extended ASCII** (**EASCII** or **high ASCII**) character encodings are eight-bit or larger encodings that include the standard seven-bit ASCII characters, plus additional characters.

128 additional characters is not enough to cover all purposes, all languages, or even all European languages, so the emergence of *many* proprietary and national ASCII-derived 8-bit character sets was inevitable.
(see encodings – iconv)

Μετατροπή Τύπων (Type Conversion)

- Για να μπορεί ένας υπολογιστής να εκτελεί αριθμητικές πράξεις, οι **τελεσταίοι (operands)** πρέπει να έχουν **το ίδιο μέγεθος** (δηλ., αριθμό bits) αλλά και να **αποθηκεύονται με τον ίδιο τρόπο.**
- **Έμμεση / Αυτόματη Μετατροπή Τύπου (Implicit Conversion)**
  - π.χ., 16-bit `short` + 32-bit `int`, ο μεταγλωττιστής θα διευθετήσει έτσι ώστε ο `short` να γίνει 32 bits πριν την πρόσθεση.
  - Περίπλοκοι κανόνες, λόγω ύπαρξης πολλών τύπων στη C.
  - **Στα πλαίσια του μαθήματος να ΑΠΟΦΕΥΓΕΤΑΙ στο μέγιστο.**
- **Ρητή Μετατροπή Τύπου (Explicit Conversion / Casting)**
  - Η Μετατροπή γίνεται από τον προγραμματιστή.
  - Π.χ., int x = (int) percentage + 1;
  - **Στα πλαίσια του μαθήματος να χρησιμοποιείται ΠΑΝΤΑ**

Μετατροπή Τύπων (Type Conversion)

- Η ρητή μετατροπή τύπων είναι κάποτε απαραίτητη για αποφυγή προβλημάτων υπερχείλισης:

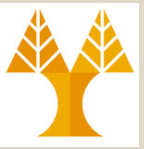
```
long i;
int j = 1000000;
i = j * j; /*overflow may occur as j*j=10^12 > 2*10^9 */
```

- Χρησιμοποιώντας το cast x64 διορθώνει το πρόβλημα:

```
i = (long) j * j; // καλύτερα i = ((long) j) * j;
```

- Η έκφραση `i = (long) (j * j);  /*** WRONG ***/`
  - δεν είναι σωστή, εφόσον η υπερχείλιση θα είχε ήδη δημιουργηθεί κατά την ανάθεση του αποτελέσματος
- 3 τρόποι διαίρεσης με το ίδιο αποτέλεσμα:
  - `(float) dividend / divisor`
  - `dividend / (float) divisor;`
  - `(float) dividend / (float) divisor;`

Ρητή Μετατροπή Τύπων (Casting) Παραδείγματα

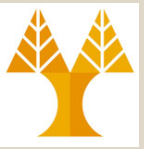


```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    // Character to Integer
    int i = (int)('9' - '0') ;
    printf("%d\n",i);

    // String to Integer
    int j = (int)atoi("9") ;
    printf("%d\n",j);

    // Float/Double to Integer
    int k = (int)9.0 ;
    printf("%d\n",k);

    //  Character to String
    char sa[2];
    sa[0] = '9';
    sa[1] = '\0';
    printf("%s\n", sa);

    // Integer to String
    // not on Linux GCC solution:
    // itoa(9, sa, 10); // number, buffer, base
    sprintf(sa, "%d", 9);
    printf("%s\n", sa);
}
```

Prints:

9
9
9
9
9

Δήλωση (Νέων) Τύπων (Type Definitions)



- Για να δηλώσουμε νέους τύπους κάνουμε χρήση του typedef (***type definition):***

```
typedef unsigned char BYTE;
```

Compiler token
(Σέβεται Εμβέλεια Ορισμού)

- Η χρήση γίνεται πλέον με τον γνωστό τρόπο:

```
BYTE flag;   /* same as unsigned char flag; */
```

- Η δημιουργία νέων τύπων είναι καλή για λόγους **μεταφερσιμότητας (portability)** του κώδικα
  - `int i = 100000;` δημιουργεί υπερχείλιση σε 16-bit Η/Υ
  - Παράδειγμα Χρήσης: `typedef unsigned long int size_t;`
    (κοιτάξετε το <stdint.h> για μερικές έτοιμες δηλώσεις)
- Εναλλακτικός τρόπος είναι με χρήση του `#define` (θα αποφεύγεται στα πλαίσια του μαθήματος) :

```
#define BYTE unsigned char
```

Preprocessor token,
Καθολική εμβέλεια μόνο ☹

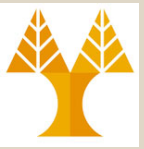
Ο Τελεστής sizeof (`sizeof` operator)

- **sizeof(type-name):** ένας μοναδιαίος τελεστής ο οποίος επιστρέφει το μέγεθος ενός τύπου (ή μεταβλητής, σταθεράς, έκφρασης i+j) σε bytes.
  - `sizeof(char)` = 1, για τους άλλους τύπους ανατρέξετε στον πίνακα της διαφάνειας 3.4-3.5 που συζητήσαμε νωρίτερα.
- Κάνοντας χρήση του **γράμματος `z`** (σε C99) μπορούμε να τυπώσουμε το μέγεθος ανεξαρτήτως πλατφόρμας (π.χ., αντί "%ld"):

```
printf("Size of int: %zu\n", sizeof(int));
```

- Ο μεταγλωττιστής ΔΕΝ μπορεί να εντοπίσει ορθά:
  - Α) το μέγεθος ενός **δυναμικού πίνακα (σε C99),** τον όποιο θα δούμε αργότερα, εφόσον ο πίνακας μπορεί να αλλάζει μέγεθος.
  - Β) το μέγεθος της περιοχής που δείχνει ένας δείκτης, τον όποιο θα δούμε αργότερα, **`char *c; sizeof(c)` είναι λάθος**

Περιεχόμενο Διάλεξης



- **Βασικοί Τύποι – Basic Types (Κεφ. 7)**
  - int, char, float, double, signed/unsigned, x86/x64
  - Υπερχείλιση τύπου, μετατροπή τύπου (έμμεση, ρητή), ορισμός τύπου, τελεστής sizeof
- **Πίνακες – Arrays (Κεφ. 8)**
  - Δήλωση, Αρχικοποίηση, sizeof,
  - Πολυδιάστατοι & μεταβλητού-μήκους πίνακες (C99),
- **Συναρτήσεις - Functions (Κεφ. 9)**
  - Ορισμός, Κλήση, Πρότυπα, Ορίσματα, Πέρασμα Τιμών Δια Τιμής, Κοινά Λάθη, Πέρασμα Πινάκων δια-διεύθυνσης, static, return, exit,
  - Αναδρομή.

Μονοδιάστατοι Πίνακες (Arrays)

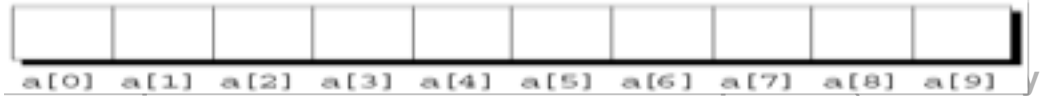
- **Πίνακας (*array*)** είναι μια βασική δομή δεδομένων η οποία περιέχει τιμές δεδομένων, του ίδιου **τύπου**, σε **συνεχόμενες διευθύνσεις μνήμης**.

```
/* Reverses a series of numbers */
#include <stdio.h>
#define N 10

int main(void) {
  int a[N], i;

  printf("Enter %d numbers: ", N);
  for (i = 0; i < N; i++)
    scanf("%d", &a[i]);

  printf("In reverse order:");
  for (i = N - 1; i >= 0; i--)
    printf(" %d", a[i]);
  printf("\n");
  return 0;
}
```

Χρήση: Όμοια με JAVA και άλλες γλώσσες που είδατε μέχρι στιγμής!

Με &, διότι ζητούμε καταχώρηση στη διεύθυνση μνήμης του a[i]

Χωρίς &, διότι ζητούμε απλά εκτύπωση της τιμής a[i].



Μονοδιάστατοι Πίνακες (Arrays)

- Αρχικοποίηση (τα υπόλοιπα στοιχεία είναι 0) :

```
int a[10] = {1, 2, 3, 4, 5, 6};
```

- Αρχικοποίηση σε μηδέν:

```
int a[10] = {0};
```

- Παράληψη μεγέθους (το βρίσκει ο μεταγλωττιστής):

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- Επιλεκτική αρχικοποίηση (σε C99):

```
int a[15] = {[2] = 29, [9] = 7, [14] = 48};
```

- Εύρεση μεγέθους πίνακα:

```
for (i = 0; i < sizeof(a) / sizeof(a[0]); i++) a[i] = 0;
```

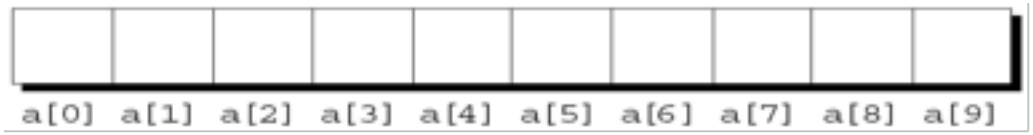
40 4

```
ή #define SIZE ((int) (sizeof(a) / sizeof(a[0])))
  for (i = 0; i < SIZE; i++) a[i] = 0; // Αντικατάσταση
SIZE με δήλωση σταθεράς (από προ-επεξεργαστή)
```

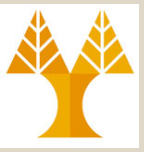
- Σταθεροί πίνακα: `const int a[] = {1,5,6,9};`

Υπερχείλιση Πίνακα (Array Overflow)

- Για λόγους ευελιξίας, θα δούμε πολύ αργότερα τον πραγματικό λόγο, η C επιτρέπει **αναφορά σε σημεία ενός** πίνακα **εκτός του εύρους** του,
  - **π.χ., int a[10]; a[10]=1;**
  - Σε αυτή την περίπτωση, η ακέραια τιμή “1” θα γραφτεί στα επόμενα 4 bytes που ακολουθούν τα bytes που έχουν δεσμευτεί στον πίνακα.
  - Εάν αυτό γίνεται από λάθος, τότε ενδέχεται να προκληθούν λάθη εκτέλεσης εφόσον θα γίνουν override άλλες τιμές που βρίσκονται αποθηκευμένες μετά τον πίνακα



Πολυδιάστατοι Πίνακες (Matrix)



- **Πολυδιάστατοι Πίνακες (*array*):** Όμοια με JAVA και άλλες γλώσσες που είδατε μέχρι στιγμής!

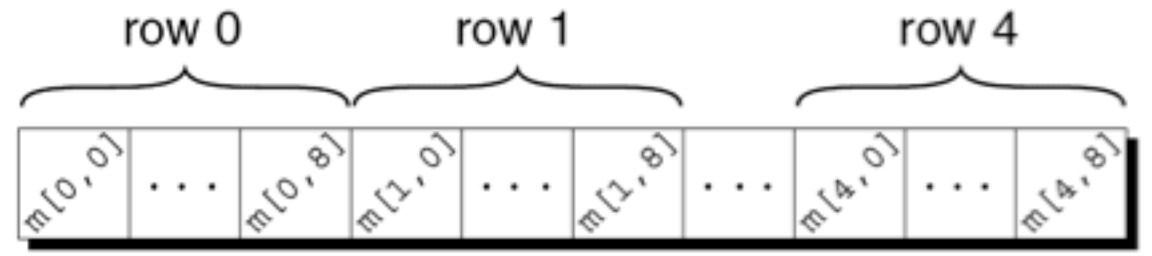
```
int m[5][9]; // 5 γραμμές και 9 στήλες
```

Αρχικοποίηση: `m[5][9]={{0}};`

Σάρωση: `&m[i][j];` Εκτύπωση: `m[i][j]`

Παρόλο που αναπαριστάται ο πίνακας ως δυσδιάστατος, στην πραγματικότητα τα στοιχεία του πίνακα είναι αποθηκευμένα σε **συνεχόμενες διευθύνσεις μνήμης**

- Η γλώσσα C αναπαριστά τα στοιχεία σε ***row-major order,*** πρώτα η γραμμή 0, μετά η γραμμή 1 κτλ.



`m[i][j] = m[i*rowsize + j]`

Πολλά παράδειγμα στη Διάλεξη 6!

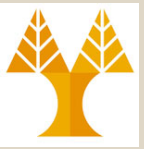
Μεταβλητού-μήκους Πίνακες (Variable-Length Arrays – C99 μόνο)

- **Μεταβλητού-μήκος Πίνακας (*VLA*):** Πίνακας του οποίου το μέγεθος υπολογίζεται κατά την εκτέλεση (αντί κατά τη μεταγλώττιση)!
  - Υποστηρίζεται μόνο σε C99.
- **Παράδειγμα**

```
int i, n;
printf("How many numbers do you want to reverse? ");
scanf("%d", &n);
int a[n];   /* C99 only - length of array depends on n */
```

- Στα πλαίσια του μαθήματος απαγορεύεται η χρήση μεταβλητού μεγέθους πινάκων (ασκήσεις, εξετάσεις, εργαστήρια, κτλ.) !!!
  - Εφόσον για παιδαγωγικούς λόγους θέλουμε να γίνεται η διαχείριση μνήμης (δέσμευση / αποδέσμευση) από εσάς, τους προγραμματιστές!

Περιεχόμενο Διάλεξης

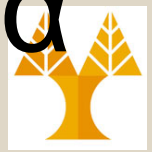


- **Βασικοί Τύποι – Basic Types (Κεφ. 7)**
  - int, char, float, double, signed/unsigned, x86/x64
  - Υπερχείλιση τύπου, μετατροπή τύπου (έμμεση, ρητή), ορισμός τύπου, τελεστής sizeof
- **Πίνακες – Arrays (Κεφ. 8)**
  - Δήλωση, Αρχικοποίηση, sizeof,
  - Πολυδιάστατοι & μεταβλητού-μήκους πίνακες (C99),
- **Συναρτήσεις - Functions (Κεφ. 9)**
  - Ορισμός, Κλήση, Πρότυπα, Ορίσματα, Πέρασμα Τιμών Δια Τιμής, Κοινά Λάθη, Πέρασμα Πινάκων δια-διεύθυνσης, static, return, exit,
  - Αναδρομή.

Συναρτήσεις (Functions)

- **Συναρτήσεις (functions*):*** **Όμοια** με JAVA και άλλες γλώσσες που είδατε μέχρι στιγμής, αλλά και με **ιδιαιτερότητες**, τις οποίες θα δούμε στη συνέχεια!
  - Χρησιμοποιούνται για **περιορισμό της εμβέλειας μεταβλητών**, **αφαιρετικότητα** και ευρύτερα, για **δομημένο προγραμματισμό.**.
  - Στην **ερχόμενη διάλεξη** θα διεισδύσουμε μέσα στην **ανατομία** ενός προγράμματος **υπό εκτέλεση** και εκεί θα καταλάβουμε καλύτερα πως προκύπτουν οι **ιδιαιτερότητες**.
  - Στο **παρόν στάδιο**, ας επικεντρωθούμε στην **απλή χρήση συναρτήσεων** όπως κάναμε στο ΕΠΛ131.

Συναρτήσεις: Απλό Παράδειγμα (Functions: Example)



Ένα απλό παράδειγμα:

```
/* Prints a countdown */

#include <stdio.h>

void print_count(int n)
{
  printf("T minus %d and counting\n", n);
  // return 0; Επιστρέφεται: void, int, float, double,
   (type *) δείκτης αλλά όχι πίνακας.
}

int main(void)
{
  int i;

  for (i = 10; i > 0; --i)
    print_count(i);

  return 0;
}
```

Τιμή επιστροφής (τίποτα)

Όρισμα (εάν δεν υπήρχε θα βάζαμε void εάν ήταν σταθερά τότε const int n)

Κλήση Συνάρτησης

Ορισμός Συναρτήσεων (Function Definitions)

- **Σειρά Ορισμού Συναρτήσεων:** Γενικά, ο ορισμός μιας συνάρτησης είναι καλό να γίνεται **πριν την κλήση της**, για λόγους συμβατότητας με παλαιοτέρα πρότυπα, εάν και η C99 επιτρέπει και το αντίθετο.
  - **Πρότυπα Συναρτήσεων (Function Declarations or Prototypes):** Στην επόμενη διαφάνεια θα δούμε πως τα πρότυπα συναρτήσεων χρησιμεύουν στην ενημέρωση του μεταγλωττιστή για την ύπαρξη συναρτήσεων.
- **Εμβέλεια Μεταβλητών:** Κάθε μεταβλητή ορίζεται στα πλαίσια του σώματος της συνάρτησης
  - εκτός από static μεταβλητές που θα δούμε αργότερα
- **Κενό Σώμα Συνάρτησης:** Καλή τακτική για δημιουργία σκελετού προγράμματος

```
void print_pun(void)
{           }
```

Πρότυπα Συναρτήσεων (Function Prototypes)

Με το **πρότυπο** γνωρίζει ο **μεταγλωττιστής** ότι κάπου στο **πρόγραμμα ορίζεται η συνάρτηση** (σάρωση πάνω=>κάτω)

Είναι προαιρετικά, μόνο ο τύπος χρειάζεται για να

```
#include <stdio.h>

double average(double a, double b);    /* DECLARATION */

int main(void)
{
  double x, y, z;

  printf("Enter three numbers: ");
  scanf("%lf%lf%lf", &x, &y, &z);
  printf("Average of %g and %g: %g\n", x, y, average(x, y));
  printf("Average of %g and %g: %g\n", y, z, average(y, z));
  printf("Average of %g and %g: %g\n", x, z, average(x, z));

  return 0;
}

double average(double a, double b)     /* DEFINITION */
{
  return (a + b) / 2;
}
```

64-bit

g (exponential | fixed decimal ανάλογα με το μέγεθος).

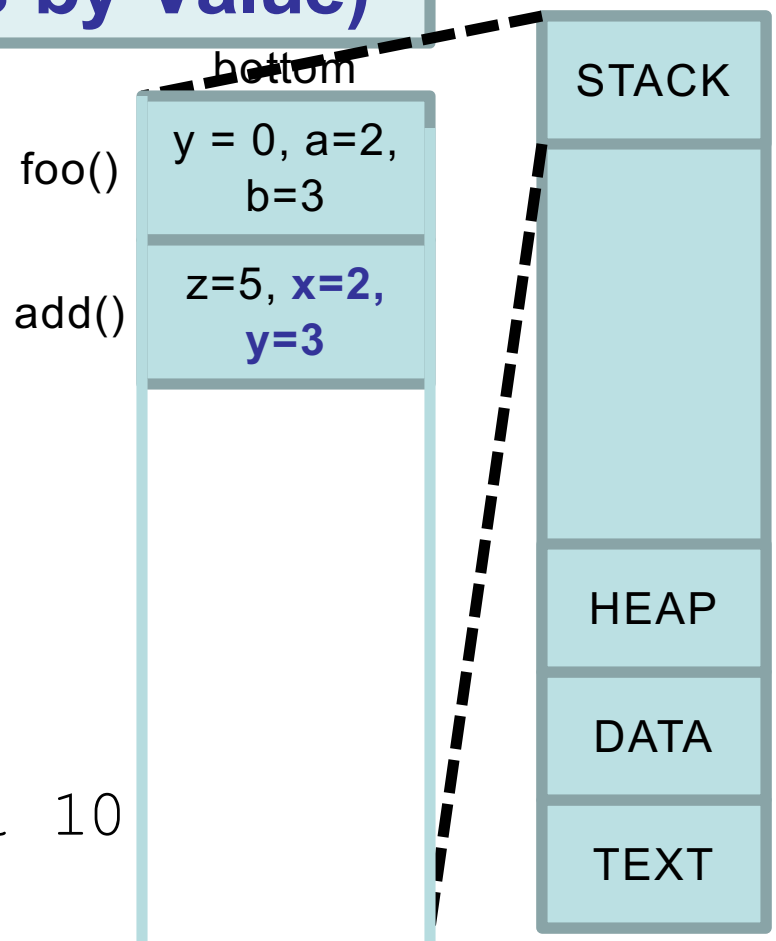
Ορίσματα Συναρτήσεων (Function Arguments)

- Στη C, τα ορίσματα πέρανε σε μια **συνάρτηση δια-τιμής (*passed by value*):**
  - Όταν καλείται η συνάρτηση κάθε **όρισμα αντιγράφεται** (COPY) σε **περιοχή μνήμης** που «ανήκει» στη συνάρτηση (**στοίβα προγράμματος**)
    - Περισσότερες λεπτομέρειες στην επόμενη διάλεξη!
  - **Πλεονέκτημα:** Όταν τερματίσει η συνάρτηση την εκτέλεση της, αυτός ο χώρος επιστρέφεται στο Λειτουργικό Σύστημα (**αυτόματες μεταβλητές – automatic variables**).
  - **Μειονέκτημα:** Πρέπει να εξάγουμε ότι θέλουμε από την συνάρτηση μέσω return (σε πρώτο στάδιο) και σε κατοπινό στάδιο (από διάλεξη 5), μέσω δεικτών **πριν το τέλος της συνάρτησης**.
  - Η επόμενη διαφάνεια δείχνει διαγραμματικά την λογική πίσω από το πέρασμα μεταβλητής.

Πέρασμα Μεταβλητών Δια Τιμής (Call-by-Value)

- **Πέρασμα-Δια-Τιμής (Pass-by-Value)**

```
int add(int x, int y) {
   int z = 5;
   return x+y+z;
}

int foo() {
   int y=0, a=2, b=3;
   y = add(a,b);
   printf("%d", y); // δίνει 10
   return 0;
}
```



(Αυτόματη) Μετατροπή Ορισμάτων Argument Conversions

- Στη C επιτρέπεται το πέρασμα μεταβλητών, υπό προϋποθέσεις, χωρίς να συμπίπτει ο τύπος. **Παράδειγμα:**

```
#include <stdio.h>

int main(void) {
  double x = 3.0;
  printf("Square: %d\n", square(x));
  return 0;
}

int square(int n) {
  return n * n;
}
```

Λάθος ☹ Αποτέλεσμα: π.χ.,1

Απροσδιόριστη Συμπεριφορά!

```
#include <stdio.h>
int square(int);

int main(void) {
  double x = 3.0;
  printf("Square: %d\n", square(x));
  return 0;
}
int square(int n) {
  return n * n; }
```

Ορθό ☺ Αποτέλεσμα: 9

- Αυτό είναι επικίνδυνο και ΠΡΕΠΕΙ να αποφεύγεται.
  - Αυτό επειδή κατά την κλήση της `square`, ο μεταγλωττιστής, που σαρώνει το πρόγραμμα από **πάνω προς τα κάτω** δεν έχει **δει**:
    - α) το πρότυπο της square (για να γνωρίζει τι uprgrade float->double->long double, int->uint->long int-> ulint πρέπει να κάνει), **ΟΥΤΕ**
    - β) τον ορισμό της συνάρτηση.
  - Για αποφυγή τέτοιων προβλημάτων κάνουμε πάντα CASTING
    - `printf("Square: %d\n", square((int)x));`
  - Ακόμη καλύτερα, δίνουμε το πρότυπο της συνάρτησης και CASTING

Ορίσματα Πινάκων (Array Arguments)

- Εάν το όρισμα είναι ένα 1D πίνακας, τότε το **μέγεθος** του μπορεί να παραμείνει **απροσδιόριστο.**
  - Ωστόσο, πρέπει να παρέχουμε **το μήκος** του πίνακα εάν επιθυμούμαι να **επεξεργαστούμε τα στοιχεία** του.
    - Εάν δώσουμε λάθος μήκος θα έχουμε λάθος λειτουργία!

Μέγεθος Μήκος

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

- **Πρότυπο Συνάρτησης**

```
int sum_array(int [], int);
```

- Κλήση Συνάρτησης

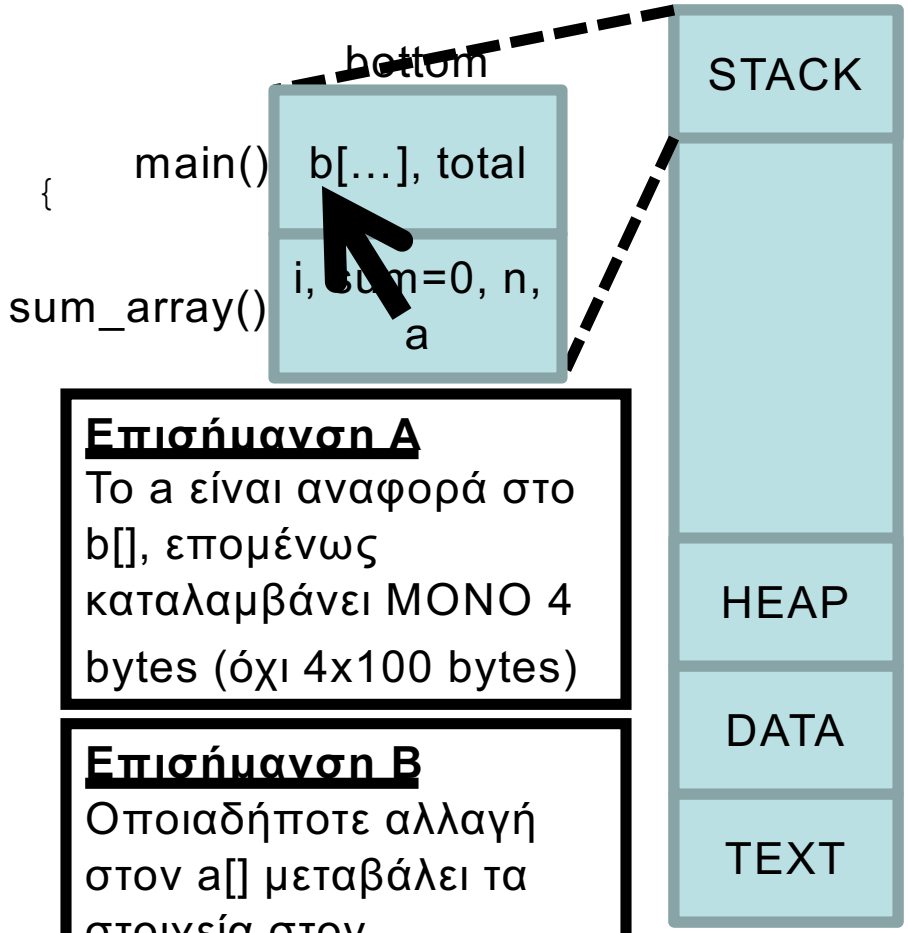
```
#define LEN 100
int main(void)
{
  int b[LEN], total;
  …
  total = sum_array(b, LEN);
  …
}
```

Σημειώστε ότι δεν υπάρχει το «&», αυτό εφόσον ο πίνακας περνά στη συνάρτηση δια-διεύθυνσης (όχι δια-τιμής)

Πέρασμα Μεταβλητών Δια Διεύθυνσης (Call-by-Reference)

- **Πέρασμα Δια-Διεύθυνσης (Pass-by-Reference)**

```
#define LEN 100

int sum_array(int [], int);

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

int main(void)
{
  int b[LEN], total;
  …
  total = sum_array(b, LEN);
  …
}
```



Επισήμανση Α
Το a είναι αναφορά στο b[], επομένως καταλαμβάνει ΜΟΝΟ 4 bytes (όχι 4x100 bytes)

Επισήμανση Β
Οποιαδήποτε αλλαγή στον a[] μεταβάλει τα στοιχεία στον πραγματικό πίνακα b[]

Ορίσματα Πολυδιάστατων Πινάκων (Matrix Arguments)

- Εάν η παράμετρος είναι **πολυδιάστατος πίνακας** τότε το **μήκος** τότε το μέγεθος της **πρώτης διάστασης** μπορεί να παραβλεφθεί, **ΑΛΛΑ** όχι των **υπολοίπων διαστάσεων**.
- **Παράδειγμα**

```
#define LEN 10

int sum_two_dimensional_array(int a[][LEN], int n){
  int i, j, sum = 0;

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

- Με τη χρήση Πίνακα Δεικτών, τους οποίους θα δούμε στη διάλεξη 6, μπορούμε να μη-δηλώνουμε το μέγεθος των υπόλοιπων διαστάσεων (για ευελιξία)

Επιστροφή Αποτελέσματος (Η εντολή `return`)

- Κάθε συνάρτηση πρέπει να **επιστρέφει** κάποια **τιμή εξόδου** στην συνάρτηση που **κάνει την κλήση**.
- **Αυτό γίνεται παραδοσιακά για 2 λόγους:**
  - **A. Επιστροφή Αποτελέσματος** για «Μαθηματικές» Συναρτήσεις:
    - π.χ., int add(2,3) => επιστρέφει το αποτέλεσμα της πράξης.
  - **B. Επιβεβαίωση** (error code) ότι μια πράξη εκτελείται ορθά, για πιο σύνθετες συναρτήσεις (οι οποίες αφορούν παράγοντες εκτός προγράμματος, π.χ., μνήμη, χρήστη, δίσκο, δίκτυο, κτλ)
    - π.χ., int saveRecordToFile(int ID, char name[]), όπου πρέπει να ανοίξει το αρχείο, να καταχωρηθεί η εγγραφή, να κλείσει το αρχείο, κτλ.
    - Η συνάρτηση επιστρέφει ένα ακέραιο για να δείξει επιτυχία (0) ή αποτυχία (>0)
- Για την περίπτωση B, δημιουργείται βέβαια το πρόβλημα του πως παίρνουμε πίσω κάποιο(α) αποτελέσματα από μια συνάρτηση.
  - Αυτό θα επιλυθεί όταν δούμε στην Διάλεξη 5 πως περνάμε σε συναρτήσεις οι μεταβλητές δια-διεύθυνσης.

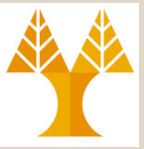
Επιστροφή Αποτελέσματος (Η εντολή `return`)

- **Αναγκαιότητα χρήσης return:** Μια void συνάρτηση δεν χρειάζεται return (void foo()) ή απλά `return;` χωρίς τιμή
  - Ωστόσο, ο τύπος επιστοφής (void) είναι υποχρεωτικός σε C99!
- **Σύνθετες Εκφράσεις Επιστροφής:**

  `π.χ., return (n >= 0 ? n : 0);`
- **Επιστροφή του main(): OK => 0 και ERROR => κάποιο** ακέραιο που δίνεται πίσω στη διεργασία-πρόγραμμα που εκτέλεσε τον πρόγραμμα, το οποίο δύναται να το χρησιμοποιήσει (π.χ., για error-handling)
  - `Από το κέλυφος: $./program ; echo $?;`

    `Τυπώνει το exit code του προγράμματος σας!`
- **exit(0):** Διακόπτει το πρόγραμμα και επιστρέφει «0» στον καλών (τη διεργασία-προγράμματος που το εκτέλεσε)
  - Να αποφεύγεται η χρήση του εκτός από το main()

Οι Μακροεντολές:
EXIT_SUCCESS / EXIT_FAILURE



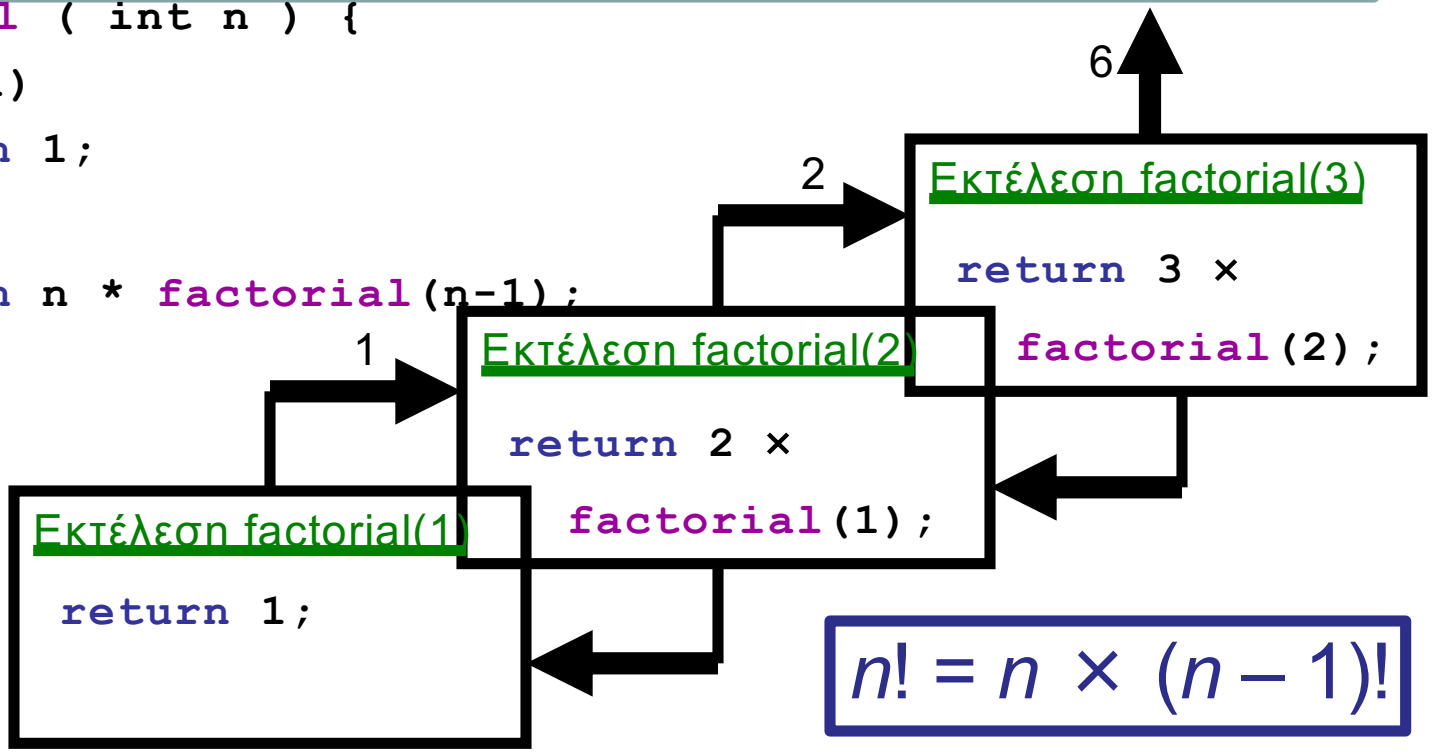
- Εφόσον η επιστροφή 0 ή 1 μπορεί να μπερδεύει, η C επιτρέπει τη χρήση δυο χρήσιμων μακροεντολών που ορίζονται στη βιβλιοθήκη **<stdlib.h>.**
  - EXIT_SUCCESS (ακέραια τιμή 0)
  - EXIT_FAILURE (ακέραια τιμή 1)
- Παράδειγμα

```
int saveRecordToFile(int ID, char name[]) {
    if (!openFile()) return EXIT_FAILURE;
    ...
    return EXIT_SUCCESS;
}
…
int main() {
    if (saveRecordToFile() == EXIT_FAILURE)
        return EXIT_FAILURE;
}
```

Αναδρομή (Recursion)

- Όπως και στη JAVA, η C επιτρέπει την κλήση αναδρομής **(recursion),** μια συνάρτηση η οποία καλεί τον εαυτό της με μια συνθήκη τερματισμού.

```
int factorial ( int n ) {
   if (n <= 1)
        return 1;
   else
        return n * factorial(n-1);
}
```



$$n! = n \times (n-1)!$$

Οι επαναληπτικές κλήσεις στοιβάζονται στη στοίβα του προγράμματος