ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Εισαγωγή: Τύποι, Πίνακες, Συναρτήσεις

(Κεφάλαια 7-8-9, KNK-2ED)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

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

Το μάθημα αυτό δομήθηκε βάση των διαλέξεων του Αναπλ. Καθηγητή Δημήτρη Ζεϊναλιπούρ

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.**

Dr. Andreas Aristidou a.aristidou@ieee.org

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.

Dr. Andreas Aristidou a.aristidou@ieee.org

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

Dr. Andreas Aristidou a.aristidou@ieee.org

Βασικοί Τύποι Δεδομένων

Τι γνωρίζαμε μέχρι σήμερα;

Μηχανές Lab (Linux, Νέα IntelMacs, κτλ.)

- 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)`

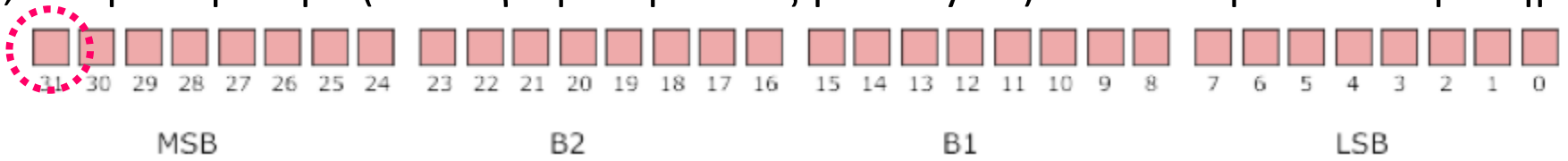
Size in bits

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

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.

0 - Εάν ο αριθμός είναι θετικός ή μηδενικός,
1 - Αν είναι αρνητικός



- 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)

- Signed Integers
  - Ο μεγαλύτερος ακέραιος 16-bit έχει τη δυαδική αναπαράσταση 0111111111111111, και ισούται με 32,767 ($2^{15} – 1$).
  - Ο μεγαλύτερος ακέραιος 32-bit είναι 01111111111111111111111111111111 και ισούται με 2,147,483,647 ($2^{31} – 1$).
- Unsigned Integers
  - Ο μεγαλύτερος ακέραιος 16-bit χωρίς πρόσημο είναι 65,535 ($2^{16} – 1$).
  - Ο μεγαλύτερος ακέραιος χωρίς πρόσημο 32-bit είναι 4,294,967,295 ($2^{32} – 1$).

Τύποι Ακέραιων Αριθμών (Integer Types)

- Ο τύπος `int` είναι συνήθως 32 bits, αλλά μπορεί να έχει και 16 bits σε παλαιότερους CPUs.
  - Το εύρος τιμών που αντιστοιχεί σε κάθε έναν από τους τύπους ακεραίων ποικίλλει από ένα μηχάνημα σε άλλο
- Οι ***Long*** ακεραίοι μπορεί να έχουν περισσότερα bits από τους απλούς ακέραιους. Αντίστοιχα οι ***short*** ακεραίοι μπορεί να έχουν λιγότερα bits.
- Τα προσδιοριστικά `long` και `short`, καθώς επίσης και τα `signed` και `unsigned`, μπορούν να συνδυαστούν με `int` για να ορίσουν τους ακόλουθους τύπους ακέραιων.
  - Μόνο έξι συνδυασμοί μπορούν να παραχθούν:

```
short int       unsigned short int
int             unsigned int
long int        unsigned long int
```

- Η σειρά των προσδιοριτών δεν έχει σημασία. Επίσης, η λέξη `int` μπορεί να απαλειφθεί (`long int` μπορεί να συντομευθεί μόνο σε `long`).

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`

Οκταδικοί και δεκαεξαδικοί αριθμοί

- Οι οκταδικοί αριθμοί χρησιμοποιούν μόνο τα ψηφία 0 έως 7.
- Κάθε θέση σε οκταδικό αριθμό αντιπροσωπεύει μια ισχύ/δύναμη του 8.
  - Ο οκταδικός αριθμός 237 αντιστοιχεί στον δεκαδικό αριθμό $2 \times 8^2 + 3 \times 8^1 + 7 \times 8^0 = 128 + 24 + 7 = 159$.
- Ένας δεκαεξαδικός (ή hex) αριθμός γράφεται με τα ψηφία 0 έως 9 συν τα γράμματα A έως F, τα οποία αντιστοιχούν στους αριθμούς από 10 έως 15, αντίστοιχα.
  - Ο δεκαεξαδικός αριθμός 1AF ισούται με την δεκαδική τιμή $1 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 256 + 160 + 15 = 431$.

Σταθερές ακεραίων

- Οι ***Decimal*** σταθερές περιέχουν ψηφία μεταξύ 0 και 9, αλλά δεν πρέπει να ξεκινούν με μηδενικό:

  `15  255  32767`

- Οι ***Octal*** σταθερές περιέχουν μόνο ψηφία μεταξύ 0 και 7 και πρέπει να ξεκινούν με μηδενικό:

  `017  0377  077777`

- Οι ***Hexadecimal*** σταθερές περιέχουν ψηφία μεταξύ 0 και 9 και γράμματα μεταξύ `a` και `f`, και πάντα να αρχίζει με `0x`:

  `0xf  0xff  0x7fff`

- Τα γράμματα σε μια δεκαεξαδική σταθερά μπορεί να είναι είτε κεφαλαία είτε με μικρά:

  `0xff  0xfF  0xFf  0xFF  0Xff  0XfF  0XFf  0XFF`

Dr. Andreas Aristidou a.aristidou@ieee.org

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.
  - Οι `long long` πρέπει να είναι τουλάχιστον 64 bits!

Dr. Andreas Aristidou a.aristidou@ieee.org

Σταθερές ακεραίων

- Για να επιβάλετε στο μεταγλωττιστή να αντιμετωπίσει μια σταθερά ως long integer, απλά τοποθετήστε στο τέλος το γράμμα L (ή l):

  `15L  0377L  0x7fffL`

- Αντίστοιχα, για να την ορίσετε σαν unsigned, τοποθετήστε το γράμμα U (ή u) :

  `15U  0377U  0x7fffU`

- Τα L και U μπορούν να συδυαστούν:

  `0xffffffffUL`

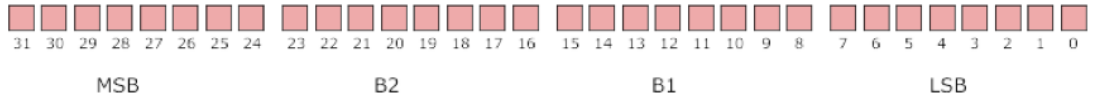
  Οι σειρά που δηλώνονται τα L και U δεν έχει σημασία.

- Ο τύπος μιας δεκαδικής σταθεράς χωρίς επίθημα (U, u, L, l, LL, ή ll) είναι η "μικρότερη" από τους τύπους `int`, `long int`, ή `long long int` που μπορεί να αντιπροσωπεύει την αξία αυτής της σταθεράς.

Dr. Andreas Aristidou a.aristidou@ieee.org

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)**
    - 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 καταβάλουν διπλανά bits στη μνήμη, σε αντίθεση με την υπερχείλιση συμβολοσειρών / πινάκων που θα δούμε αργότερα.

Παράδειγμα
Άθροιση μιας σειράς αριθμών

- Το πρόγραμμα `sum.c` αθροίζει μια σειρά ακεραίων (δες τις διαφάνειες του προηγούμενου μαθήματος).
- Ένα πρόβλημα με αυτό το πρόγραμμα είναι ότι το άθροισμα (ή ένας από τους αριθμούς εισόδου) μπορεί να υπερβεί τη μεγαλύτερη τιμή που επιτρέπεται για ένα `int`.
- Δείτε τι μπορεί να συμβεί αν το πρόγραμμα εκτελείται σε ένα μηχάνημα του οποίου οι ακέραιοι έχουν μέγεθος 16 bits :

```
This program sums a series of integers.
Enter integers (0 to terminate): 10000 20000 30000 0
The sum is: -5536
```

- Όταν εμφανίζεται υπερχείλιση με signed αριθμούς, το αποτέλεσμα είναι απροσδιόριστο.
- Το πρόγραμμα μπορεί να βελτιωθεί με τη χρήση `long` μεταβλητών.

Παράδειγμα
Άθροιση μιας σειράς αριθμών

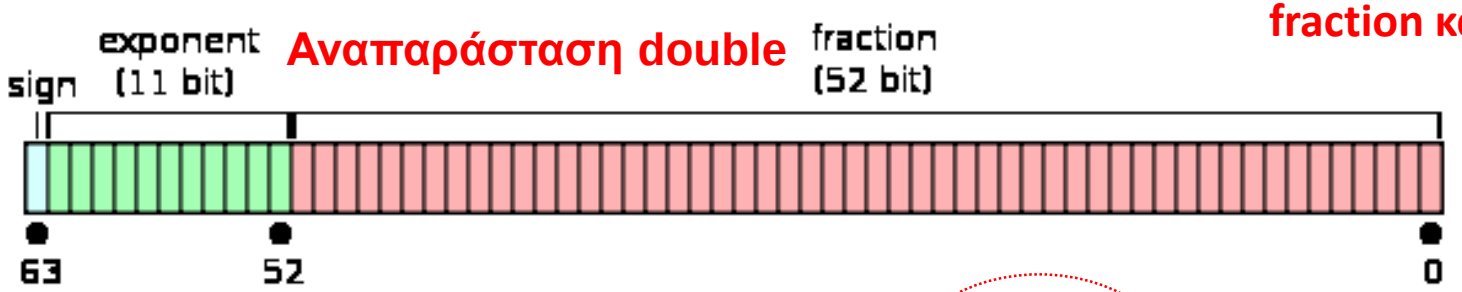
sum2.c

```
/* Sums a series of numbers (using long variables) */

#include <stdio.h>

int main(void)
{
  long n, sum = 0;

  printf("This program sums a series of integers.\n");
  printf("Enter integers (0 to terminate): ");

  scanf("%ld", &n);
  while (n != 0) {
    sum += n;
    scanf("%ld", &n);
  }
  printf("The sum is: %ld\n", sum);

  return 0;
}
```

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).

Αντίστοιχα, στα float, ο exponent είναι 8 bits, και fraction καταλαμβάνει 23 bits



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 Πραγματικών Αριθμών (Floating Types)

- Οι Floating σταθερές μπορούν να γραφτούν με διάφορους τρόπους.
- Έγκυροι τρόποι εγγραφής του αριθμού 57.0:

```
57.0  57.  57.0e0  57E0  5.7e1  5.7e+1
.57e2  570.e-1
```

- Εξ ορισμού, οι floating σταθερές φυλάσσονται σαν double.
- Για να υποδείξετε ότι θέλετε μόνο μία ακρίβεια στη σταθερά, βάλτε το γράμμα `F` (ή `f`) στο τέλος (π.χ. `57.0F`).
- Για να υποδείξετε ότι θέλετε `long double` σταθερά, βάλτε το γράμμα `L` (ή `l`) στο τέλος (π.χ. `57.0L`).

Τύποι Πραγματικών Αριθμών (Floating Types)

- Τα %e, %f, και %g χρησιμοποιούνται για την ανάγνωση και την εγγραφή αριθμών κινητής υποδιαστολής (float) μονής ακρίβειας.
- Κατά την ανάγνωση μιας τιμής τύπου `double`, βάλτε το γράμμα `l` μπροστά από τα `e`, `f`, ή `g`:

```
double d;

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

  - *Σημείωση:* Χρησιμοποιήστε το `l` μόνο στη `scanf`, όχι στην `printf`.
- Για την `printf`, τα `e`, `f`, και `g` μπορούν να χρησιμοποιηθούν τόσο για `float` όσο και για `double`.
- Κατά την ανάγνωση ή την εγγραφή μιας τιμής τύπου `long double`, βάλτε το γράμμα `L` μπροστά από τα `e`, `f`, ή `g`.

Dr. Andreas Aristidou a.aristidou@ieee.org

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.
  - Οι τιμές του τύπου `char` μπορεί να ποικίλλει από έναν υπολογιστή σε άλλο, επειδή διαφορετικοί υπολογιστές μπορεί να χρησιμοποιούν διαφορετικούς χαρακτήρες.
- 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**;

Ο Πίνακας ASCII (πρώτοι 128 χαρακτήρες – 7bit)

- American Standard Code for Information Interchange

ASCII Code Chart

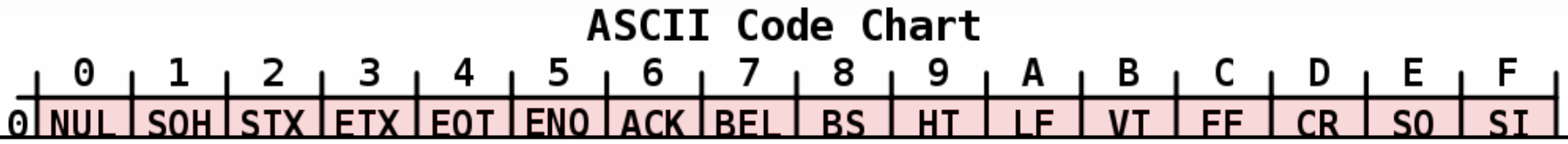
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Οι 128 ASCII χαρακτήρες, συμπεριλαμβανομένου των μη εκτυπώσιμων χαρακτήρων (αναπαρίστανται με συντομογραφία).

Dr. Andreas Aristidou a.aristidou@ieee.org

Ο Πίνακας ASCII (πρώτοι 128 χαρακτήρες – 7bit)

- American Standard Code for Information Interchange



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 are 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)

Latin-1 that provides the characters necessary for Western European and many African languages

Οι 128 ASCII χαρακτήρες, συμπεριλαμβανομένου των μη εκτυπώσιμων χαρακτήρων (αναπαρίστανται με συντομογραφία).

Τύποι Χαρακτήρων (Character Types)

- Μια μεταβλητή τύπου `char` μπορεί να αντιστοιχιστεί σε οποιονδήποτε μεμονωμένο χαρακτήρα:

```
char ch;

ch = 'a';   /* lower-case a */
ch = 'A';   /* upper-case A */
ch = '0';   /* zero         */
ch = ' ';   /* space        */
```

- Παρατηρήστε ότι οι σταθερές χαρακτήρων περικλείονται σε μονά εισαγωγικά και όχι σε διπλά εισαγωγικά.

Τύποι Χαρακτήρων (Character Types)

- Όταν ένας χαρακτήρας εμφανίζεται σε έναν υπολογισμό, τότε η C χρησιμοποιεί την ακέραια τιμή του.
- Έστω τα ακόλουθα παραδείγματα, τα οποία είναι σε ASCII:

```
char ch;
int i;

i = 'a';        /* i is now 97   */
ch = 65;        /* ch is now 'A' */
ch = ch + 1;    /* ch is now 'B' */
ch++;           /* ch is now 'C' */
```


Dr. Andreas Aristidou a.aristidou@ieee.org

Τύποι Χαρακτήρων (Character Types)

- Το γεγονός ότι οι χαρακτήρες έχουν τις ίδιες ιδιότητες με τους αριθμούς έχει κάποια πλεονεκτήματα.
- Για παράδειγμα, είναι εύκολο να γράψετε μια εντολή `for` των οποίων η μεταβλητή ελέγχου διέρχεται από όλα τα κεφαλαία γράμματα:

  ```
  for (ch = 'A'; ch <= 'Z'; ch++) …
  ```

- Μειονεκτήματα της χρήσης των χαρακτήρων ως αριθμών:
  - Μπορεί να οδηγήσει σε σφάλματα που δεν θα αναγνωριστούν από τον μεταγλωττιστή.
  - Επιτρέπει εκφράσεις χωρίς νόημα, όπως `'a' * 'b' / 'c'`.
  - Μπορεί να παρεμποδίσει τη φορητότητα του προγράμματος (διαφορετικά πρότυπα).

Χαρακτήρες διαφυγής

- Μια πλήρης λίστα με τους χαρακτήρες διαφυγής:

Name	***Escape Sequence***
Alert (bell)	`\a`
Backspace	`\b`
Form feed	`\f`
New line	`\n`
Carriage return	`\r`
Horizontal tab	`\t`
Vertical tab	`\v`
Backslash	`\\`
Question mark	`\?`
Single quote	`\'`
Double quote	`\"`

Οι χαρακτήρες διαφυγής μπορούν να αναπαραστήσουν ένας συγκεκριμένο χαρακτήρα χρησιμοποιώντας την οκταδική ή δεκαεξαδική τιμή του χαρακτήρα.
Π.χ.
Μια οκταδική ακολουθία διαφυγής αποτελείτε από το `\` ακολουθούμενη από τον οκταδικό αριθμό 3 ψηφίων, πρδ. `\33` ή `\033`.
Μια δεκαεξαδική ακολουθία ***διαφυγής*** αποτελείτε από το `\x` ακολουθούμενη από τον δεκαεξαδικό αριθμό `\x1b` ή `\x1B`.
Το `x` πρέπει αν είναι σε μικρά γράμματα, αλλά τα δεκαεξαδικά ψηφία μπορεί να είναι τόσο κεφαλαία όσο και μικρά.

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.
  - Αν προσθέσουμε ένα `int` και ένα `float`, ο μεταγλωττιστής θα διευθετήσει έτσι ώστε ο `int` να γίνει `float`.
  - 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.
  - Η στρατηγική πίσω από τις συνήθεις αριθμητικές μετατροπές: Μετατρέψτε τους τελεσταίους στο "στενότερο" τύπο που θα φιλοξενήσει με ασφάλεια και τις δύο τιμές.
  - Π.χ., int x = (int) percentage + 1;
  - **Στα πλαίσια του μαθήματος να χρησιμοποιείται ΠΑΝΤΑ**

Dr. Andreas Aristidou a.aristidou@ieee.org

Μετατροπή Τύπων (Type Conversion)

- Παράδειγμα :

```
char c;
short int s;
int i;
unsigned int u;
long int l;
unsigned long int ul;
float f;
double d;
long double ld;

i = i + c;     /* c is converted to int                */
i = i + s;     /* s is converted to int                */
u = u + i;     /* i is converted to unsigned int       */
l = l + u;     /* u is converted to long int           */
ul = ul + l;   /* l is converted to unsigned long int */
f = f + ul;    /* ul is converted to float             */
d = d + f;     /* f is converted to double             */
ld = ld + d;   /* d is converted to long double        */
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Μετατροπή Τύπων κατά την ανάθεση

```
char c;
int i;
float f;
double d;

i = c;   /* c is converted to int   */
f = i;   /* i is converted to float */
d = f;   /* f is converted to double */
```

- Η ανάθεση αριθμού κινητής υποδιαστολής σε μια μεταβλητή ακεραίων ρίχνει το κλασματικό τμήμα του αριθμού:

```
int i;

i = 842.97;    /* i is now 842 */
i = -842.97;   /* i is now -842 */
```

- Η ανάθεση μιας τιμής σε μια μεταβλητή ενός «στενότερου» τύπου θα δώσει ένα ακαθόριστο αποτέλεσμα, ειδικά αν η τιμή βρίσκεται εκτός εύρους τιμών του τύπου της μεταβλητής:

```
c = 10000;    /*** WRONG ***/
i = 1.0e20;   /*** WRONG ***/
f = 1.0e100;  /*** WRONG ***/
```

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;`

Cast

Το cast έχει την ακόλουθη μορφή

(*type-name*) *expression*

Όπου *type-name* καθορίζει τον τύπο στον οποίο θα πρέπει να μετατραπεί η παράσταση.

Dr. Andreas Aristidou a.aristidou@ieee.org

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 (operator) `sizeof`

- **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.

Dr. Andreas Aristidou a.aristidou@ieee.org

Μονοδιάστατοι Πίνακες (Arrays)

Πίνακας (*array*) είναι μια βασική δομή δεδομένων η οποία περιέχει τιμές δεδομένων (***elements)***, 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].

Dr. Andreas Aristidou a.aristidou@ieee.org

Μονοδιάστατοι Πίνακες (Arrays)

- Αρχικοποίηση (τα υπόλοιπα στοιχεία είναι 0) :

```
int a[10] = {1, 2, 3, 4, 5, 6};
/* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

- Αρχικοποίηση σε μηδέν:

```
int a[10] = {0};
/* initial value of a is {0, 0, 0, 0, 0, 0, 0, 0, 0, 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};
int a[15] = {0, 0, 29, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 48};
```

- Εύρεση μεγέθους πίνακα:

40 bytes 4 bytes

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

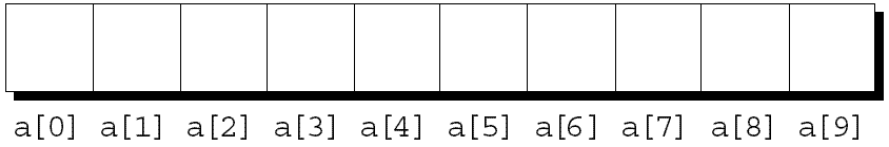
→ A loop that clears the array

```
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};`

Η χρήση macro για να ορίσετε το μήκος μιας συστοιχίας/πίνακα είναι μια εξαιρετική πρακτική:

```
#define N 10
…
int a[N];
```



An array with *n* elements is indexed from 0 to *n* – 1, not 1 to *n*

Μονοδιάστατοι Πίνακες (Arrays)

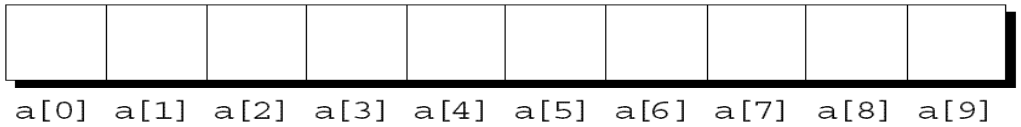
- Πολλά προγράμματα έχουν βρόγχους `for` η εργασία των οποίων είναι να εκτελέσει κάποια λειτουργία σε κάθε στοιχείο ενός πίνακα.
- Παράδειγμα:

```
for (i = 0; i < N; i++)
  a[i] = 0;                /* clears a */

for (i = 0; i < N; i++)
  scanf("%d", &a[i]);      /* reads data into a */

for (i = 0; i < N; i++)
  sum += a[i];             /* sums the elements of a */
```


Dr. Andreas Aristidou a.aristidou@ieee.org

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



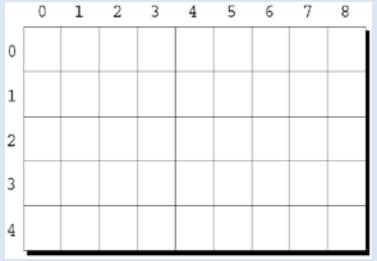
Πολυδιάστατοι Πίνακες (Matrix)

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

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

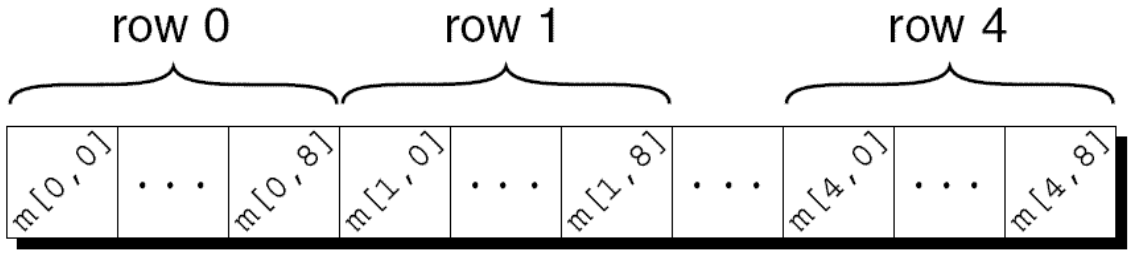
Αρχικοποίηση: `m[5][9]={{0}};`

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!

Πολυδιάστατοι Πίνακες (Matrix)

- Αρχικοποίηση:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

- Αν η αρχικοποίηση δεν φτάνει για όλα τα στοιχεία του πίνακα, τότε τα εναπομείνοντα στοιχεία θα έχουν τιμή μηδέν.
- Αν η αρχικοποίηση δεν φτάνει για όλα τα στοιχεία μιας γραμμής του πίνακα, τότε τα εναπομείνοντα στοιχεία της θα έχουν τιμή μηδέν.

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1},
               {0, 1, 0, 1, 1, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

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.

Dr. Andreas Aristidou a.aristidou@ieee.org

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 **επόμενη 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.

Παράδειγμα

Συναρτήσεις (Functions)

```
/* 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 10 φορές

Παράδειγμα

Συναρτήσεις (Functions)

- Όταν μια συνάρτηση δεν έχει παραμέτρους, η λέξη `void` τοποθετείται σε παρενθέσεις μετά το όνομα της συνάρτησης:

```
void print_pun(void)
{
  printf("To C, or not to C: that is the question.\n");
}
```

- Για να καλέσετε μια λειτουργία χωρίς παραμέτρους, γράφουμε το όνομα της συνάρτησης, ακολουθούμενο από παρενθέσεις:

```
print_pun();
```

  Οι παρενθέσεις *πρέπει* να είναι παρούσες.

  Πάντα χρήση ελληνικού ερωτηματικού στο τέλος

- Η κλήση μιας `non-void` συνάρτησης παράγει μια τιμή που μπορεί να αποθηκευτεί σε μια μεταβλητή, να εκτυπωθεί, κτλ:

```
avg = average(x, y);
```

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)

```
#include <stdio.h>

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;
}
```

- Όταν ο μεταγλωττιστής βλέπει για πρώτη φορά την κλήση της `average` στη `main`, δεν έχει καμιά πληροφορία για αυτή
- Αντί να παράγει ένα μήνυμα σφάλματος, το πρόγραμμα μεταγλώττισης υποθέτει ότι η `average` επιστρέφει μια τιμή `int`.
- Λέμε ότι ο μεταγλωττιστής έχει δημιουργήσει μια ***implicit declaration*** της συνάρτησης

Πρότυπα Συναρτήσεων (Function Prototypes)

```
#include <stdio.h>

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;
}
```

- Το πρόγραμμα μεταγλώττισης αδυνατεί να ελέγξει ότι περνάμε τον σωστό αριθμό ορισμάτων, και ότι τα ορίσματα έχουν τον κατάλληλο τύπο.
- Όταν συναντά τον ορισμό της συνάρτησης αργότερα στο πρόγραμμα, ο μεταγλωττιστής παρατηρεί ότι ο τύπος της συνάρτησης είναι στην πραγματικότητα double και όχι Int και έτσι εμφανίζεται ένα μήνυμα σφάλματος.

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

```
#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).

Dr. Andreas Aristidou a.aristidou@ieee.org

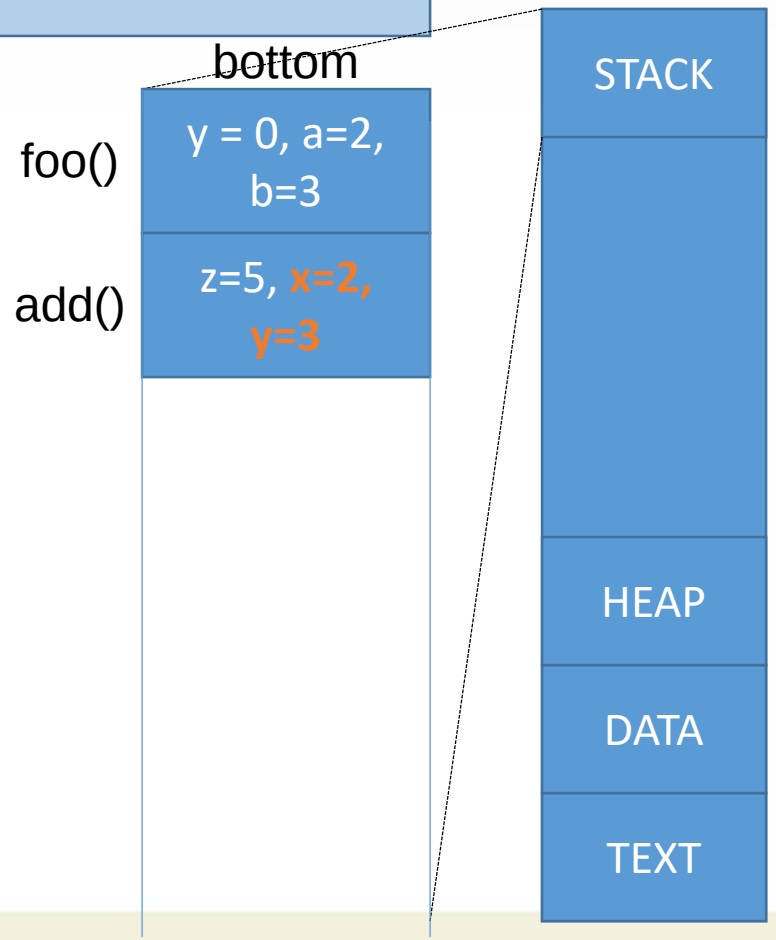
Ορίσματα Συναρτήσεων (Function Arguments)

- Στη C, τα ορίσματα περνάνε 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 επόμενο στάδιο (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;
}
```



Argument Conversions

(Αυτόματη) Μετατροπή Ορισμάτων

- 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 επιθυμούμε 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)

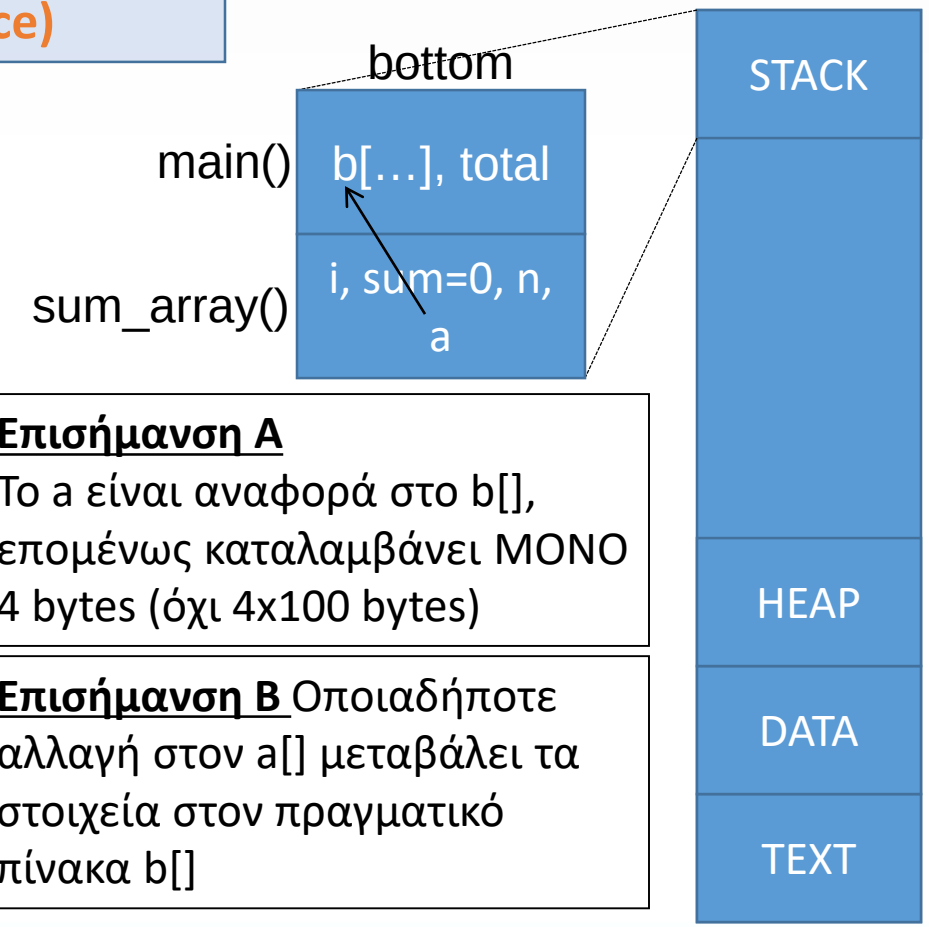
Dr. Andreas Aristidou a.aristidou@ieee.org

Call-by-Reference

Πέρασμα Μεταβλητών Δια Διεύθυνσης

- **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[]

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:**
  - **Α. Επιστροφή Αποτελέσματος** για «Μαθηματικές» Συναρτήσεις:
    - π.χ., int add(2,3) => επιστρέφει το αποτέλεσμα της πράξης.
  - **Β. 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 Β, 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

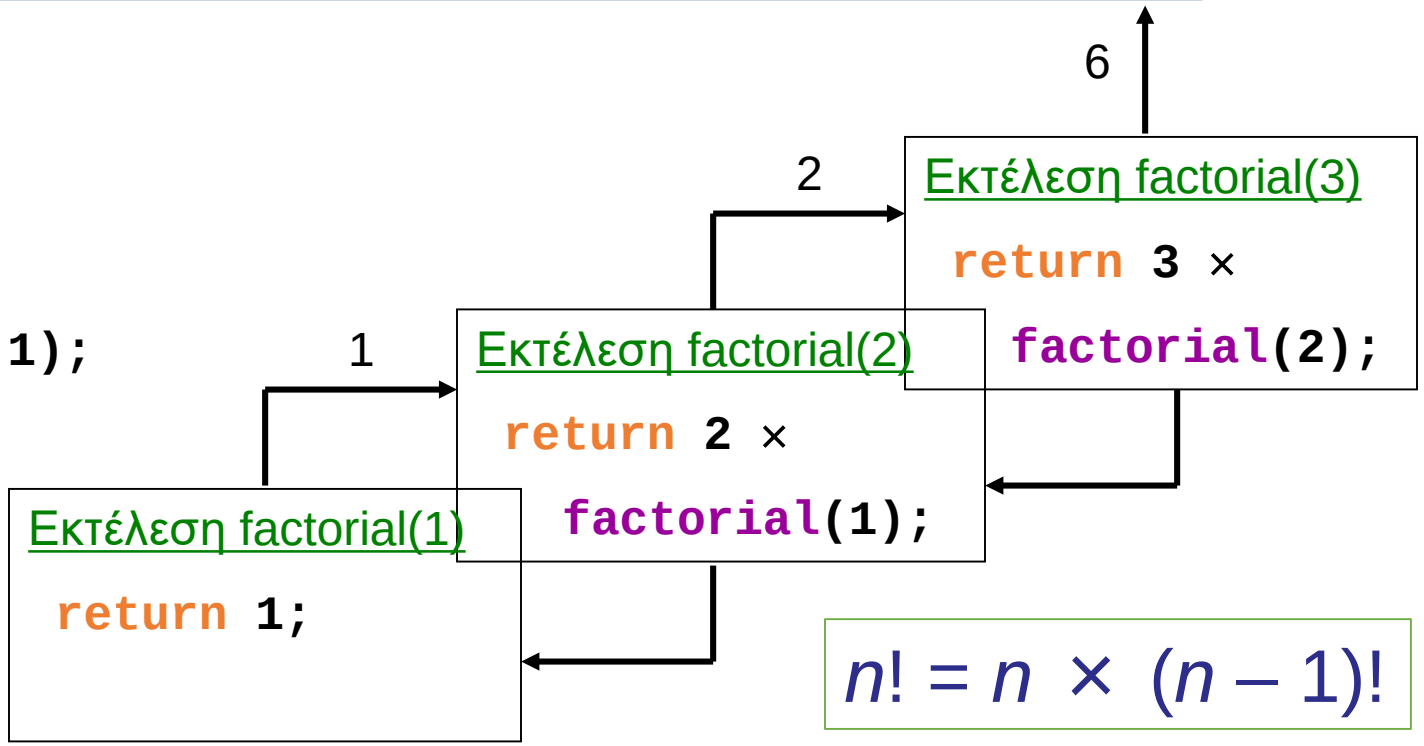
- Εφόσον η επιστροφή 0 ή 1 μπορεί να μπερδεύει, η C επιτρέπει τη χρήση δυο χρήσιμων μακροεντολών που ορίζονται στη βιβλιοθήκη **<stdlib.h>**.
  - EXIT_SUCCESS (ακέραια τιμή 0)
  - EXIT_FAILURE (ακέραια τιμή 1)
- Παράδειγμα

```
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

Αναδρομή (Recursion)

- Για να δείτε πώς λειτουργεί η αναδρομή, ας δούμε το ακόλουθο παράδειγμα

```
i = factorial(3);
```

`factorial(3)` finds that 3 is not less than or equal to 1, so it calls

`factorial(2)`, which finds that 2 is not less than or equal to 1, so it calls

`factorial(1)`, which finds that 1 is less than or equal to 1, so it returns 1, causing

`factorial(2)` to return 2 × 1 = 2, causing

`factorial(3)` to return 3 × 2 = 6.

Dr. Andreas Aristidou a.aristidou@ieee.org