Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

6α

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Διάλεξη #6

Παρασκευή, 16 Απριλίου 2021

Παναγιώτης Παύλου

c-programming@allos.gr

Αναπαράσταση κειμένων

Αναπαράσταση μεμονωμένου χαρακτήρα και κειμένου

Κωδικοποίηση χαρακτήρων

Οι χαρακτήρες κειμένου στον υπολογιστή παριστάνονται ως αριθμοί βάσει κάποιας αντιστοίχισης η οποία ονομάζεται κωδικοποίηση (encoding) του κειμένου. Παραδοσιακά **κάθε χαρακτήρας αντιστοιχεί σε έναν αριθμό (και αντίστροφα)**.

Η πιο γνωστή κωδικοποίηση κειμένου χρησιμοποιεί κωδικούς αριθμούς, οι οποίοι στο δυαδικό παριστάνονται με 7bits (άρα τιμές 0 – 127). Ονομάζεται **ASCII** (δείτε και εδώ) και ουσιαστικά είναι ένας πίνακας αντιστοίχισης για τους χαρακτήρες του Αγγλικού αλφαβήτου, των αριθμητικών ψηφίων και κάποιων συμβόλων προς τους κωδικούς τους.

Επειδή το κάθε byte έχει 8bits οι υπόλοιπες 128 τιμές (128-255) που δεν ορίζονται από το ASCII μπορούν να περιέχουν διάφορους χαρακτήρες. Το ποιοι χαρακτήρες είναι αυτοί εξαρτάται από την παραλλαγή της κωδικοποίησης που θα επιλεγεί. Συνήθως υπάρχει μία παραλλαγή για κάθε γλώσσα.

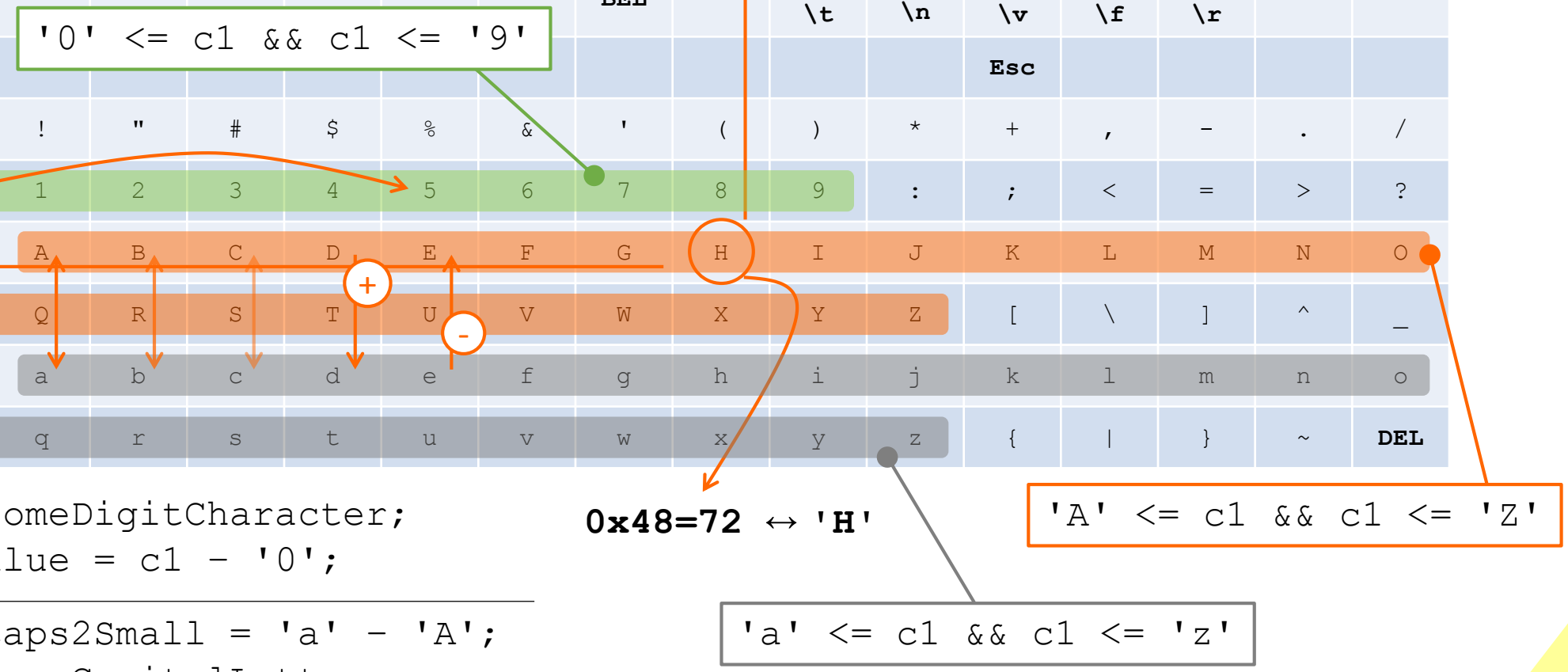
Άρα για την αναπαράσταση κατά ASCII σε κάθε χαρακτήρα αρκούν 8bits = 1byte, γι'αυτό και ο τύπος δεδομένων του ακεραίου των 8bits στη C ονομάζεται **`char`**.

Στις νέες πολυγλωσσικές κωδικοποιήσεις (encodings) υπάρχει δυνατότητα συνδυασμού περισσότερων bytes για την αναπαράσταση των χαρακτήρων. Έτσι μπορεί μία κωδικοποίηση να καλύψει περισσότερους από 256 διαφορετικούς χαρακτήρες, πράγμα απαραίτητο για πολυγλωσσικά κείμενα. Τέτοιες κωδικοποιήσεις είναι οι UTF-8, UTF-16, κλπ

Στο μάθημα θα περιοριστούμε σε κείμενα με Αγγλικούς χαρακτήρες και στην κωδικοποίηση ASCII με 7bits.

Πίνακας ASC-II

bits 4-6	bits 0-3	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0								BEL		HTab \t	Enter \n	Vtab \v	FF \f	CR \r		
0001	1												Esc				
0010	2	Space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL




```
› char c1 = someDigitCharacter;
  int digitValue = c1 - '0';
```

```
› int offsetCaps2Small = 'a' - 'A';
  char c2 = someCapitalLetter;
  char c3 = c2 + offsetCaps2Small;
```

Ομάδες χαρακτήρων – ctype

- **Ψηφία / Digits**
  Τα αριθμητικά ψηφία 0 1 2 3 4 5 6 7 8 9
  `int isdigit(int c)`
- **Δεκαεξαδικά ψηφία / Hexadecimal digits**
  Τα αριθμητικά ψηφία και οι χαρακτήρες a-f
  0 1 2 3 4 5 6 7 8 9 A a B b C c D d E e F f
  `int isxdigit(int c)`
- **Πεζά γράμματα / Lowercase letters**
  Οι πεζοί χαρακτήρες του Αγγλικού αλφαβήτου
  a b c d e f g h i j k l m n o p q r s t u v w x y z
  `int islower(int c)`
- **Κεφαλαία γράμματα / Uppercase letters**
  Οι κεφαλαίοι χαρακτήρες του Αγγλικού αλφαβήτου
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  `int isupper(int c)`
- **Γράμματα ή Αλφαβητικοί χαρακτήρες / Letters ή Alphabetic characters**
  Τα κεφαλαία και τα πεζά γράμματα μαζί
  `int isalpha(int c)`
- **Αλφαριθμητικοί χαρακτήρες / Alphanumeric characters**
  Τα γράμματα μαζί με τα (αριθμητικά) ψηφία
  `int isalnum(int c)`
- **Σημεία στίξης / Punctuation characters**
  Οι χαρακτήρες
  ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
  `int ispunct(int c)`

```
0 1 2 3 4 5 6 7 8 9

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

ΠΡΟΣΟΧΗ!
Χρησιμοποιήστε τις επιστρεφόμενες τιμές ως λογικές μεταβλητές.

- **Γραφικοί χαρακτήρες / Graphical characters**
  Οι αλφαριθμητικοί χαρακτήρες μαζί με τα σημεία στίξης. Δηλαδή όλοι οι χαρακτήρες οι οποίοι έχουν οπτική αναπαράσταση (εμφανίζουν κάτι στην οθόνη).
  `int isgraph(int c)`
- **Κενοί χαρακτήρες / Space ή Whitespace characters**
  Χαρακτήρες που έχουν επίδραση στο αποτέλεσμα αλλά δεν έχουν ορατό περιεχόμενο. Πιο συγκεκριμένα είναι το κενό ( ), το οριζόντιο tab (\t), το κατακόρυφο tab (\v), αλλαγή γραμμής (\n), επιστροφή (\r) και form feed (\f).
  `int isspace(int c)`

Αυτοί είναι και οι χαρακτήρες που χρησιμοποιούνται για την ελεύθερη σύνταξη ενός προγράμματος C

- **Εκτυπώσιμοι χαρακτήρες / Printable characters**
  Οι αλφαριθμητικοί χαρακτήρες και οι κενοί μαζί.
  `int isprint(int c)`
- **Χαρακτήρες ελέγχου / Control characters**
  Χαρακτήρες με κωδικό 0 (\000) ως και 31 (\037) και ο 127 (\177)
  `int iscntrl(int c)`

Κείμενα

Τα κείμενα στη C είναι απλά πίνακες χαρακτήρων. Η μόνη διαφορά είναι ότι το επόμενο στοιχείο από τον τελευταίο χαρακτήρα θα πρέπει να περιέχει τον κωδικό ASCII **0**, αντί να χρειάζεται να είναι γνωστό το μήκος τους!

Άρα το κείμενο Hello θα μπορούσε να γραφεί όπως ξέρουμε ως πίνακας:

```
char greeting[] = { 'H', 'e', 'l', 'l', 'o', 0 };
```

NULL terminated string

Ένωση σταθερών κειμένων κατά το compilation ως εξής:
`"one,""two""",three"`
που ταυτίζεται με το
`"one,two,three"`

Αλλά αυτό είναι και κάπως "κουραστικό". Γι'αυτό το ισοδύναμα γράφεται:

```
char greeting[] = "Hello";
```

το οποίο τοποθετεί από μόνο του το πρόσθετο στοιχείο με κωδικό 0. Δηλαδή ο πίνακας όπως και πριν είναι με 6 στοιχεία.

Η χρήση του μηδενικού στο τέλος του κειμένου δεν είναι σπατάλη μνήμης ενός χαρακτήρα, αλλά είναι οικονομία αφού για την αποθήκευση του μήκους ενός κειμένου εν γένει θα χρησιμοποιούνταν μία ακέραια μεταβλητή τουλάχιστον 2 ή 4 bytes.

Εάν βέβαια χρειαστεί να γνωρίζουμε το μήκος του κειμένου, τότε θα χρειαστεί να υπολογιστεί.

```
int calculateLength(char text[]) {
    int length;
    for (length = 0; text[length] != 0; length++) {
        ;
    }
    return length;
}
```

Εμβέλεια μεταβλητών

Περισσότερες δυνατότητες

global εμβέλεια μεταβλητών 1/2

Ο πρώτος μηχανισμός είναι η χρήση μεταβλητών έξω από όλες τις συναρτήσεις και – κατά συνέπεια – έξω από όλα τα block εντολών. Αυτές οι μεταβλητές ονομάζονται global (καθολικές) και μπορούν να χρησιμοποιηθούν από οποιοδήποτε σημείο του κώδικα, σε αντίθεση με τις τοπικές (local) μεταβλητές που γνωρίζαμε έως τώρα.

Οι εφαρμογές τέτοιων μεταβλητών είναι η αποθήκευση κοινόχρηστης πληροφορίας όπως είναι ρυθμίσεις του προγράμματος, γενικοί μετρητές ή σε άλλη περίπτωση κάποια πληροφορία που θα εξαρτάται από το εκάστοτε πρόβλημα που επιλύεται.

```
int someGlobalCounter = 0;
void func1() {
    someGlobalCounter++;
    ...
}
void func2() {
    someGlobalCounter--;
    ...
}
```

global εμβέλεια μεταβλητών 2/2

Προφανώς η χρήση των μεταβλητών επιτρέπεται από το σημείο της δήλωσής τους και κάτω. Αυτό παίζει ρόλο όταν δηλώσετε μια global μεταβλητή ανάμεσα στις συναρτήσεις.

Η αρχική τιμή (όταν αναγράφεται) δίνεται πριν την εκτέλεση της πρώτης εντολής της main και πρέπει να είναι τιμή γνωστή κατά το compilation, δεν μπορεί να είναι το αποτέλεσμα της κλήσης μιας συνάρτησης. Ενναλακτικά η αρχικοποίησή μιας global μεταβλητής μπορεί να γίνεται σε όποιο σημείο του κώδικα κρίνει κατάλληλο ο προγραμματιστής.

Τέλος οι μεταβλητές αυτές, όταν ένα πρόγραμμα είναι γραμμένο σε περισσότερα του ενός αρχεία κώδικα, είναι διαθέσιμες σε όλα αυτά. Αυτό καθιστά υπεύθυνο τον προγραμματιστή ώστε να μην υπάρχει σύμπτωση ονομάτων μεταξύ των διαφόρων αρχείων.

Στατικές μεταβλητές

Ο δεύτερος μηχανισμός είναι η δήλωση μεταβλητής ως στατική με τη λέξη κλειδί `static`. Η επίδραση της «διευκρίνισης» static εξαρτάται από το αν η μεταβλητή είναι global ή local.

Για τις τοπικές μεταβλητές η επίδραση είναι η εξής: Μετά το τέλος του block στο οποίο είναι δηλωμένη η μεταβλητή, αυτή δεν διαγράφεται από τη μνήμη, παρότι δεν είναι προσβάσιμη από τον κώδικα. Έτσι όταν η εκτέλεση του κώδικα επιστρέψει στο ίδιο block η μεταβλητή είναι και πάλι προσβάσιμη και περιέχει την προηγούμενη τιμή της. Φυσικά όταν δίνεται αρχική τιμή σε αυτή κατά τη δήλωσή της, η τιμή αποδίδεται μόνο στην πρώτη εκτέλεση της συνάρτησης (ή του block).

Για τις global μεταβλητές η δήλωσή της ως static, απλά περιορίζει την εμβέλειά τους στο αρχείο στο οποίο δηλώνονται, κάνοντάς τες ασφαλείς ως προς «συνωνυμίες» με global μεταβλητές άλλων αρχείων.

Εφαρμογή

```
double useThis(int number) {
    static int count=0;
    static int sum=0;
    count++;
    sum += number;
    return sum/(double)count;
}
```

Εάν μπει όμως σε ξεχωριστό αρχείο του project τότε...

```
static int count=0;
static int sum=0;

double useThis(int number) {
    count++;
    sum += number;
    return sum/(double)count;
}
void resetThis() {
    count = 0;
    sum = 0;
}
double getThis() {
    return sum/(double)count;
}
```

- ✓ Αυτοπεριεχόμενο
- ✗ Δεν μηδενίζεται
- ✗ Η μέση τιμή δίνεται μία φορά
- ✓ Προστατευμένη τιμή

- ✓ Δέσμευση ονομάτων global μεταβλητών
- ✓ Μηδενίζεται
- ✓ Συνεχή πρόσβαση στη μέση τιμή
- ✓ Οι τιμές είναι εκτεθειμένες

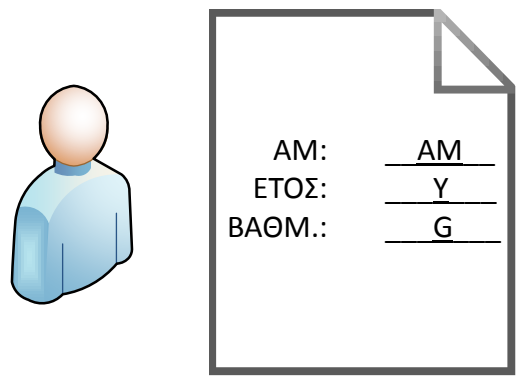
Δομές δεδομένων

Οργάνωση πληροφορίας

Συσχέτιση της πληροφορίας 1/2

Όταν περιγράφουμε ένα πρόβλημα με μαθηματικό τρόπο, δηλαδή όταν το μοντελοποιούμε, υπάρχουν ποσότητες οι οποίες σχετίζονται μεταξύ τους πιο στενά από τις υπόλοιπες. Για να τις παραστήσουμε όλες τις ποσότητες χρησιμοποιούνται απλές μεταβλητές, οπότε δεν γίνεται εμφανής αυτός ο διαχωρισμός.

Χρειαζόμαστε λοιπόν έναν τρόπο που να αναδεικνύει τη συσχέτισή τους. Για παράδειγμα για να περιγράψουμε έναν **φοιτητή** και τον βαθμό του σε ένα μάθημα, ας υποθέσουμε ότι αρκεί:

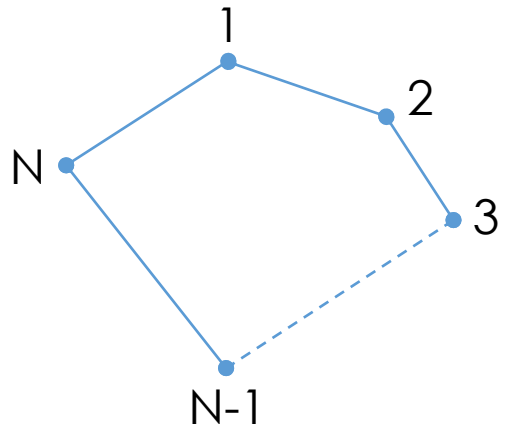


- Το όνομα της σχολής **S**
- Ο τίτλος του μαθήματος **T**
- Ο αριθμός μητρώου του φοιτητή **ΑΜ**
- Το έτος εισαγωγής **Υ** και
- Ο βαθμός στο μάθημα **G**

Όμως τα ΑΜ, Υ και G έχουν στενότερη σχέση μεταξύ τους απ'ότι με τις υπόλοιπες μεταβλητές.

Συσχέτιση της πληροφορίας 2/2

Επίσης για να περιγράψουμε ένα πολύγωνο στο επίπεδο χρειαζόμαστε:



- Το πλήθος των κορυφών του (N) και
- Τις συντεταγμένες τους, (x1,y1), (x2,y2), (x3,y3), κ.ο.κ

Όμως η σχέση του κάθε x με το αντίστοιχο y (ιδιαίτερα αφού λογικά τα x θα έχουν δηλωθεί ως πίνακας x[0], x[1], x[2],... και τα y ομοίως y[0], y[1], y[2],...) **δεν γίνεται εμφανής**.

Η κατάσταση «επιδεινώνεται» σε αυτό το παράδειγμα επειδή, παρατηρώντας τις μεταβλητές του (δηλαδή τους δύο πίνακες x και y), δεν γίνεται καθόλου εμφανές ότι οι δύο πίνακες επιβάλλεται να έχουν το ίδιο πλήθος στοιχείων. Αυτό φαίνεται μόνο από τη συμπεριφορά του κώδικα προς αυτά!

Δήλωση δομών δεδομένων (struct) 1/2

Η λύση που παρέχει η C για τέτοιες περιπτώσεις είναι η κατασκευή μίας ονοματισμένης ομάδας μεταβλητών με τη χρήση της λέξης κλειδί **`struct`** η οποία γράφεται για τα δύο παραδείγματα ως ακολούθως:

```
struct Student {
    int AM;
    int Y;
    double G;
};
struct Student someStudent;
```

Πεδία (fields) ή **Μέλη** (members) οποιουδήποτε τύπου δεδομένων

```
struct Point {
    double x;
    double y;
};
struct Point points[N];
```

Ισχύουν οι γενικοί κανόνες ονομασίας.

Λειτουργεί ως **τύπος δεδομένων**

ή

```
typedef struct Student {
    int AM;
    int Y;
 double G;
} StudentInfo;

StudentInfo someStudent;
```

```
typedef struct Point {
    double x;
    double y;
} Point2D;


Point2D points[N];
```

Δήλωση δομών δεδομένων (struct) 2/2

Κάθε δομή δεδομένων μπορεί να έχει οσαδήποτε ονοματισμένα μέλη (πεδία) οποιουδήποτε τύπου δεδομένων.

Για παράδειγμα μπορεί εκτός από αριθμητικά που είδαμε, να έχει ως πεδία:

- πίνακες & κείμενα
- άλλες δομές δεδομένων

Δεν επιτρέπεται όμως να περιέχει δομές του ίδιου τύπου.

Μια τέτοια αναδρομική δήλωση θα δημιουργούσε αυτόματα απαίτηση για άπειρη μνήμη με την δήλωση της πρώτης μεταβλητής αυτού του τύπου!

```
struct myDate {
    int monthDay;
    int monthNum;
    int year;
}


struct Person {
    char firstName[100];
    char lastName[100];
    struct myDate birthday;
    struct Person mother;
    int heightInCm;
    double weightInKgr;
}
```

Χρήση δομών δεδομένων

Η χρήση μιας δομής δεδομένων γίνεται όπως παρουσιάζεται στον διπλανό κώδικα.

Κάθε πεδίο ή μέλος δεν χρησιμοποιείται αυτόνομα,

...αλλά μόνο σε συνδυασμό με το όνομα της **μεταβλητής** του τύπου struct (όχι του ονόματος της δομής), ενωμένα με τον **τελεστή «τελεία»** .

...και ο συνδυασμός αυτός λειτουργεί όπως μία **οποιαδήποτε** μεταβλητή.

Όμως η μεταβλητή του τύπου struct (εδώ η p) μπορεί να χρησιμοποιηθεί αυτόνομα.

Επίσης πριν από τη δήλωση έστω και μίας μεταβλητής του εκάστοτε τύπου struct (π.χ. struct Point ή Point2D) δεν έχει δημιουργηθεί καμία μεταβλητή και κατά συνέπεια δεν καταλαμβάνεται καθόλου μνήμη.

Αντίστροφα με κάθε δήλωση μίας τέτοιας μεταβλητής (π.χ. εδώ η p), δημιουργούνται μία μεταβλητή για κάθε μέλος (πεδίο). Έτσι για παράδειγμα αφού κάθε double καταλαμβάνει 8bytes

```
typedef struct Point {
    double x;
    double y;
} Point2D;

Point2D points[N];

int quadrant(Point2D p) {
    if (p.x > 0) {
        if (p.y > 0) {
            return 1;
        } else {
            return 4;
        }
    } else {
        if (p.y > 0) {
            return 2;
        } else {
            return 3;
        }
    }
}
```

0 bytes

(8+8)*N bytes

p.y

Δήλωση νέων τύπων δεδομένων - typedef

Όταν χρησιμοποιείται η γραφή με το typedef πρέπει να είναι ξεκάθαρα τα ακόλουθα:

- Απαιτείται η δήλωση του ονόματος του νέου τύπου δεδομένων μετά το κλείσιμο του άγκιστρου και πριν το ; αλλιώς πρόκειται για λάθος που ο compiler το επισημαίνει απλώς ως προειδοποίηση (warning)

```
typedef struct Point {
    double x;
    double y;
} Point2D ;

Point2D points1[N];
```

- Η χρήση του δεν αναιρεί τη **παράλληλη** και **ισοδύναμη** χρήση του struct Point,

```
struct Point points2[N];
```

- Εάν (εκ παραδρομής) παραληφθεί η λέξη κλειδί `typdef`, τότε το λεκτικό που ακολουθεί το κλείσιμο του αγκίστρου της struct (εδώ το `planePoint1`) θεωρείται απλά ως μεταβλητή του τύπου `struct PointOnPlane`, η οποία «έτυχε» να δηλώνεται ταυτόχρονα με τη δομή δεδομένων.

```
struct PointOnPlane {
    double x;
    double y;
} planePoint1 ;
```

> Άρα συντακτικά είναι αποδεκτό, ενώ λογικά είναι λάθος!

Παράδειγμα

Η παρακάτω συνάρτηση επιστρέφει το κέντρο βάρους ενός πολυγώνου με N πλευρές.

```
typedef
struct Point {
    double x;
    double y;
}
Point2D;

Point2D calcCenterPoint(int N, Point2D points[N]) {
    Point2D center;
    center.x = 0.0;
    center.y = 0.0;
    for (int i = 0; i < N; ++i) {
        center.x += points[i].x;
        center.y += points[i].y;
    }
    center.x /= N;
    center.y /= N;
    return center;
}
```

Δήλωση της δομής «σημείο» με συντεταγμένες x και y τύπου double

Ορισμός τύπου δεδομένων Point2D (ώστε να μην γράφουμε συνέχεια "struct Point")

Πλήθος κορυφών του πολυγώνου

Πίνακας με τα σημεία των κορυφών του πολυγώνου

Δήλωση της μεταβλητής που θα κρατάει το κέντρο βάρους του πολυγώνου. Και αυτή σημείο είναι.

Αρχική τιμή των συντεταγμένων, ώστε να αποθηκευθεί το άθροισμα.

Άθροιση των αντίστοιχων συντεταγμένων του κάθε **`points[i]`** σημείου

Διαίρεση με το πλήθος των πλευρών/κορυφών/σημείων ώστε να προκύψει η μέση τιμή

Επιστροφή του αποτελέσματος (ολόκληρο το σημείο)

Σημαντικά σημεία

Μετά από τη σημερινή διάλεξη θα πρέπει να γνωρίζετε:

- Την παράσταση μεμονωμένων χαρακτήρων κειμένου στη C
- Την παράσταση ολόκληρων κειμένων στη C
- Τη διαφορά μιας τοπικής (local) και μίας global μεταβλητής
- Τη διαφορά μιας static από μία μη στατική μεταβλητή (local & global)
- Τη δήλωση μιας δομής δεδομένων
- Τη χρήση μιας δομής δεδομένων

Ερωτήσεις?

- Διαβάστε τις σημειώσεις, διαβάστε τις διαφάνειες και δείτε τα videos **πριν** ρωτήσετε
- **Συμβουλευτείτε** τη σελίδα ερωταποκρίσεων του μαθήματος

  https://qna.c-programming.allos.gr
- **Στείλτε** τις ερωτήσεις σας πριν και μετά το μάθημα στο

  c-programming@allos.gr

- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε μαζί τον κώδικα και τα μηνύματα λάθους από το CLion ως κείμενα με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!