ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΗΜΕΙΩΣΕΙΣ ΓΙΑ ΤΟ ΜΑΘΗΜΑ

Εισαγωγή στο Διαδικαστικό Προγραμματισμό
C

Ελευθέριος Δ. Πολυχρονόπουλος
Επίκουρος Καθηγητής

ΠΑΤΡΑ 2008

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Οι σημειώσεις που έχετε στα χέρια σας γράφηκαν για τις ανάγκες του Εργαστηρίου του Μαθήματος. Το Εργαστήριο Λογισμικού είναι βασικό εργαστηριακό μάθημα του πρώτου εξαμήνου του πρώτου έτους σπουδών του τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής του Πανεπιστημίου Πατρών. Σκοπός των σημειώσεων είναι να δώσει στον πρωτοετή φοιτητή βασικές εισαγωγικές γνώσεις προγραμματισμού, με όσο το δυνατόν απλουστευμένο τρόπο. Για το σκοπό αυτό έχει επιλεγεί η γλώσσα προγραμματισμού C. Η επιλογή της C έγινε για να συνδυάσει την θεωρία στο μάθημα Εισαγωγή Λογισμικού με το, παρόν, μάθημα του εργαστηρίου.

Σκοπός του εργαστηρίου είναι να εισάγει το φοιτητή στον προγραμματισμό παράλληλα όμως με την ταυτόχρονη εξοικείωση του με τα υπολογιστικά συστήματα και ιδιαίτερα με συστήματα που το λειτουργικό τους δεν είναι «οπτικά» φιλικό, παραθυρικό, προς το χρήστη. Έτσι, επιδιώκουμε κατά τη διάρκεια διεξαγωγής του μαθήματος, ο χρήστης με τη βοήθεια των σημειώσεων και σε συνδυασμό με το ενημερωτικό σεμινάριο που προσφέρει το Υπολογιστικό Κέντρο του Τμήματος μας, να προγραμματίσει και να εξοικειώνεται σε συστήματα Unix, χωρίς να αποκλείουμε τους φοιτητές μας από την ταυτόχρονή χρήση αυτόματων παραθυρικών συστημάτων, όπως τη χρήση του visual studio της microsoft. Έχουμε λοιπόν φροντίσει στις σημειώσεις μετά από κάθε ενότητα, όπου θεωρητικά αλλά απλουστευμένα εξηγούνται οι δομές της γλώσσας C, να υπάρχουν απλά προγραμματιστικά παραδείγματα για τα οποία δίνουμε και την έξοδο τους όπως εμφανίζεται κατά την εκτέλεση τους σε περιβάλλον command line.

ΕΙΣΑΓΩΓΗ

Η C σχεδιάστηκε και υλοποιήθηκε από τον Dennis Ritchie σε έναν υπολογιστή DEC PDP-11 χρησιμοποιώντας το λειτουργικό σύστημα UNIX. Το λειτουργικό σύστημα, ο μεταγλωττιστής της C και όλα τα προγράμματα εφαρμογών του Unix είναι γραμμένα σε γλώσσα C. Η C είναι το αποτέλεσμα μιας εργασίας η οποία ξεκίνησε με μία παλαιότερη γλώσσα, την BCPL, την οποία ανέπτυξε ο Martin Richards. Η BCPL επηρέασε μια γλώσσα με όνομα B, η οποία εφευρέθηκε από τον Ken Thompson και οδήγησε στην ανάπτυξη της C στις αρχές της δεκαετίας του ’70.

Για πολλά χρόνια το μοναδικό πρότυπο της γλώσσας C ήταν αυτό που περιγραφόταν στο εγχειρίδιο « The C Programming Language » των Brian Kernighan και Dennis Ritchie (Prentice-Hall 1978). Αλλά λόγο της τεράστιας ανάπτυξης της δημοτικότητας της γλώσσας, οργανώθηκε μία επιτροπή (ANSI : American National Standardisation Institute) το 1983 για την καθιέρωση ενός προτύπου για την C. Η διαδικασία προτοτυποποίησης χρειάστηκε σχεδόν 6 χρόνια (πολύ περισσότερα από ότι θα περίμενε οποιοσδήποτε). Το πρότυπο ANSI C τελειοποιήθηκε το 1989 και τα πρώτα αντίγραφα της έγιναν διαθέσιμα στο κοινό το 1990. Το πρότυπο αυτό προσαρμόσθηκε ελαφρά το 1996. Σήμερα πια, ουσιαστικά όλοι οι μεταγλωττιστές της C συμμορφώνονται με το πρότυπο ANSI C και αυτή είναι η έκδοση της C με την οποία θα ασχοληθούμε.

Η C αναφέρεται συχνά σαν μια γλώσσα «μεσαίου επιπέδου». Πριν από την C υπήρχαν ουσιαστικά δύο τύποι γλωσσών για τον προγραμματισμό υπολογιστικών σύστημα των οι assembly γλώσσες που ήταν χαμηλού επιπέδου δηλαδή ο προγραμματιστής είχε μια συμβολική αναπαράσταση των πραγματικών εντολών μηχανή και με αυτό τον τρόπο προσπαθούσε να προγραμματίσει. Από την άλλη πλευρά οι γλώσσες υψηλού επιπέδου προσέφεραν μια άνεση στον προγραμματισμό για το λόγο του ότι ήταν εφοδιασμένες με δομές τέτοιες ώστε ο προγραμματιστής να μπορεί πιο εύκολα να καθοδηγήσει μία μηχανή. Η διαφορά βέβαια και οι λόγοι για τους οποίους υπάρχουν ακόμα οι δύσχρηστες γλώσσες μηχανής είναι γιατί καταρχήν με μια γλώσσα υψηλού επιπέδου δεν είναι δυνατός ο προγραμματισμός όλων των συστημάτων και επίσης η γλώσσες μηχανής είναι γρηγορότερες κατά πολύ από της αντίστοιχες υψηλού επιπέδου. Η C συνδυάζει με μεγάλη επιτυχία την ευχρηστία των γλωσσών υψηλού επιπέδου με την αποτελεσματικότητα των αντίστοιχων χαμηλού επιπέδου. Για το λόγο αυτό ονομάστηκε γλώσσα «μεσαίου επιπέδου». Θα μπορούσαμε να πούμε ότι μια γλώσσα χαμηλού επιπέδου είναι ποίο εύκολα κατανοητή από μία μηχανή από ότι σε έναν άνθρωπο και ανάλογα μία γλώσσα υψηλού επιπέδου είναι ποίο κατανοητή σε έναν άνθρωπο από ότι σε μία μηχανή.

1. ΕΙΣΑΓΩΓΙΚΕΣ ΕΝΝΟΙΕΣ ΓΙΑ ΤΗΝ ΓΛΩΣΣΑ C

1.1 ΑΛΦΑΒΗΤΟ

Με τον όρο ***αλφάβητο*** εννοούμε όλους τους χαρακτήρες που ο μεταγλωττιστής της γλώσσας μου μπορεί να αναγνωρίσει. Στον παρακάτω πίνακα φαίνονται όλοι αυτοί οι χαρακτήρες.

A..Z a..z	0..9	?,:,&,~,",#,',{,(,|,_,),},*,\	+,-,*,/,^,	<,>,=

Όλα τα παραπάνω και μερικά ακόμα είναι τα αποδεκτά σύμβολα από τον μεταγλωττιστή της γλώσσας C.

1.2 ΔΕΣΜΕΥΜΕΝΕΣ ΛΕΞΕΙΣ ΚΑΙ ΣΥΜΒΟΛΑ

Η γραμματική είναι οι κανόνες με τους οποίους φτιάχνουμε λέξεις και το συντακτικό οι κανόνες με τους οποίους συνδυάζονται οι λέξεις αυτές για να δημιουργήσουν προτάσεις. Στον ακόλουθο πίνακα φαίνονται οι λέξεις οι οποίες είναι δεσμευμένες από την C και εξυπηρετούν ένα συγκεκριμένο σκοπό.

auto	double	Int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Στα πλαίσια της γραμματικής της γλώσσας υπάρχουν και οι τελεστές οι οποίοι χωρίζονται σε τέσσερις μεγάλες κατηγορίες: τους **αριθμητικούς**, τους **συγκριτικούς**, τους **λογικούς** και τέλος τους **bitwise**. Οι τελεστές είναι υπεύθυνοι για την τέλεση των αντίστοιχων πράξεων.

+	-	/	*	%
<	>	<=	>=	!=
==	&&	||	!	&
|	~	<<	>>	^

Ιδιαίτερη θέση στα προγράμματα της C έχουν τα σχόλια.Πολύ σημαντικά για την δημιουργία ενός προγράμματος C είναι τα ***σχόλια***. Το σχόλια βρίσκονται ανάμεσα στους χαρακτήρες /* */ και υπάρχουν για να διευκολύνουν την αναγνωσιμότητα των προγραμμάτων. Δεν μεταβάλλουν καθόλου τον τρόπο εκτέλεσης του προγράμματος αφού αγνοούνται τελείως από τον μεταγλωττιστή της γλώσσας. Ένα παράδειγμα σχόλιου είναι το παρακάτω:

```
/* αυτό που περιέχεται μεταξύ των χαρακτήρων "/* και */" είναι ένα σχόλιο στη C */
```

Επίσης υπάρχουν και τα σχόλια μίας γραμμής που δηλώνονται με τα σύμβολά // και επεκτείνονται μέχρι το τέλος της γραμμής χωρίς να υπάρχει ανάγκη κάποιου τερματικού συμβόλου. Τα σχόλια αυτά δεν είναι ορισμένα στο πρότυπο ANSI C αλλά λόγο της ευρείας χρήσης τους μπορεί να τα συναντήσουμε σε κάποιο πρόγραμμα C.

1.3 ΜΕΤΑΒΛΗΤΕΣ

Μια μεταβλητή είναι μια επώνυμη θέση στη μνήμη, η οποία μπορεί να καταρατά διάφορες τιμές. Σχηματικά μια μεταβλητή μπορεί να δειχτεί σαν ένα γραμματοκιβώτιο το οποίο έχει το όνομα του ιδιοκτήτη και μέσα περιέχει γράμματα. Στην C οι μεταβλητές πρέπει να δηλώνονται πριν μπορέσουν να χρησιμοποιηθούν. Η δήλωση μιας μεταβλητής εξυπηρετεί έναν σημαντικό σκοπό: ***λέει στον μεταγλωττιστή τον τύπο δεδομένων που χρησιμοποιεί η μεταβλητή.*** Η C υποστηρίζει 5 βασικούς τύπους μεταβλητών όπως φαίνεται και στον παρακάτω πίνακα.

Τύπος	**Δεσμευμένη λέξη**
Χαρακτήρας	char
Προσημασμένος ακέραιος	int
Αριθμός κινητής υποδιαστολής	float
Αριθμός κινητής υποδιαστολής διπλής ακρίβειας	double
Απουσία τιμής	void

Για να δηλώσουμε μια μεταβλητή σε ένα πρόγραμμα C χρησιμοποιούμε την ακόλουθη γενική μορφή :

Όνομα_τύπου όνομα_μεταβλητής;

π.χ.

int num;

Μπορούμε επίσης να δηλώσουμε περισσότερες από μία μεταβλητές του ίδιου τύπου στην ίδια γραμμή σύμφωνα με την ακόλουθη γενική μορφή:

Όνομα_τύπου όνομα_μεταβλητής1, όνομα_μεταβλητής2 .....
όνομα_μεταβλητήςN;

π.χ.

float a,b,x,y,num1,j;

Υπάρχουν δύο κυρίως θέσεις σε ένα πρόγραμμα C στις οποίες δηλώνουμε τις μεταβλητές μας και αυτές είναι: μέσα σε μία συνάρτηση όπου και η μεταβλητή ονομάζεται ***τοπική*** (local variable) και έξω από όλες τις συναρτήσεις όπου και η μεταβλητή ονομάζεται ***γενική*** (global variable). Οι τοπικές μεταβλητές είναι γνωστές μόνο στο μπλοκ στο οποίο είναι δηλωμένες δηλαδή στο σώμα της συνάρτησης που έχουν δηλωθεί και έχουν ζωή μέχρι να τελειώσει η εκτέλεση της συνάρτησης αυτής. Οι γενικές μεταβλητές είναι γνωστές από όλες τις συναρτήσεις και έχουν διάρκεια ζωής όλο το πρόγραμμα. Συνιστάται η χρήση των τοπικών μεταβλητών πάντα γιατί οι γενικές μεταβλητές είναι εύκολο να δημιουργήσουν δυσνόητες καταστάσεις κατά την εκτέλεση του προγράμματος.

Όπως είπαμε και νωρίτερα ο λόγος ύπαρξης των μεταβλητών είναι για να συγκρατούν τιμές κάποιου συγκεκριμένου τύπου. Όταν δηλώνεται μια μεταβλητή περιέχει ***μια τυχαία τιμή*** η οποία δεν μας ενδιαφέρει εμείς και ανάλογα με το τι θέλουμε να κάνουμε θα ορίσουμε την σωστή τιμή της

μεταβλητής αυτής. Ο ορισμός αυτός γίνεται με τη χρήση της εντολής εκχώρησης που είναι το σύμβολο = και ακολουθεί την γενική μορφή:

Όνομα_μεταβλητής = έκφραση;

π.χ.

b = 1;

Όταν η εκχώρηση τιμής γίνεται κατά την δήλωση της μεταβλητής τότε λέγεται αρχικοποίηση της μεταβλητής δηλαδή:

int b=1;

Η C μας δίνει την δυνατότητα να τροποποιούμε τους βασικούς τύπους δεδομένων της με τους ***τροποποιητές τύπων*** (type modifiers) οι οποίοι είναι **long**, **short**, **signed** και **unsigned**. Οι τέσσερις αυτοί τροποποιητές μπορούν να συνδυαστούν μεταξύ τους όπως για παράδειγμα **signed long** ή **unsigned short** κ.ο.κ. Πρέπει να σημειωθεί ότι όλοι οι τύποι δεν μπορούν να τροποποιηθούν κατά τον ίδιο τρόπο. Στον παρακάτω πίνακα φαίνονται οι τροποποιήσεις που επιτρέπονται κατά το πρότυπο ANSI C.

	int	char	float	double
long	✓			✓
short	✓			
signed	✓	✓		
unsigned	✓	✓		
signed long	✓			
unsigned long	✓			
signed short	✓			
unsigned short	✓			

1.4 ΜΕΓΕΘΗ ΤΥΠΩΝ ΜΕΤΑΒΛΗΤΩΝ

Το μέγεθος κάθε μεταβλητής έχει σημαντικό ρόλο στην χωρητική πολυπλοκότητα του προγράμματος που υλοποιούμε. Ο προγραμματιστής πρέπει να κάνει τη μεγαλύτερη δυνατή οικονομία των πόρων του υπολογιστή για τον οποίο γράφει το πρόγραμμά του. ***Κάθε κατασκευαστής έχει διαφορετικές τιμές bits που αντιστοιχεί σε κάθε μεταβλητή***. Στον παρακάτω πίνακα φαίνονται οι συνεήθεις τιμές για όλες τις βασικές μεταβλητές της γλώσσας C.

Τύπος	**Τυπικό μέγεθος**	**Ελάχιστο Πεδίο τιμών**
int	16bit ή 32bit	-32,767 έως 32,767
unsigned int	16bit ή 32bit	0 έως 65,535
signed int	16bit ή 32bit	-32,767 έως 32,767
short int	16bit	-32,767 έως 32,767
long int	32bit	-2,147,483,647 έως 2,147,483,647
unsigned short int	16bit	0 έως 65,535
signed short int	16bit	-32,767 έως 32,767
unsigned long int	32bit	0 έως 4,294,967,295
signed long int	32bit	-2,147,483,647 έως 2,147,483,647
char	8bit	-127 έως 127
unsigned char	8bit	0 έως 255
signed char	8bit	-127 έως 127
double	64bit	Ακρίβεια 10 δεκαδικών ψηφίων
long double	80bit	Ακρίβεια 10 δεκαδικών ψηφίων

Όπως θα δούμε σε επόμενα κεφάλαια, το μέγεθος μίας μεταβλητής (σε bytes) δίνεται από την συνάρτηση sizeof().

1.5 ΤΕΛΕΣΤΕΣ ΚΑΙ ΜΕΤΑΤΡΟΠΕΣ ΤΥΠΩΝ(casting)

1.5.1 ΟΙ ΑΡΙΘΜΗΤΙΚΟΙ ΤΕΛΕΣΤΕΣ

Όπως το λέει και το όνομά τους οι αριθμητικοί τελεστές είναι υπεύθυνοι για την τέλεση αριθμητικών πράξεων. Οι περισσότεροι μας είναι γνωστοί από το δημοτικό. Στον παρακάτω πίνακα φαίνονται οι αριθμητικοί τελεστές.

Τελεστής	Ενέργεια
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο ακέραιας διαίρεσης

Οι τελεστές +,-,/,* μπορούν να χρησιμοποιηθούν με οποιονδήποτε από τους πέντε βασικούς τύπους της C ο τελεστής % χρησιμοποιείται **μόνο** με τον τύπο των ακεραίων και προφέρεται modulo.

Πρέπει να επισημάνουμε ότι ο συνδυασμός αριθμητικών τελεστών και αριθμών λέγεται αριθμητική έκφραση, για παράδειγμα:

a = (5 % 3) + 1;

Οι συντελεστές ++ και -- (2 ενωμένα -) ονομάζονται ***τελεστές αύξησης και μείωσης***, αντίστοιχα. Ο τελεστής αύξησης προσθέτει 1 στον τελεστέο του, ενώ ο τελεστής μείωσης μειώνει 1. Για παράδειγμα:

```
int a = 2;
a++; /*  το a τώρα έχει την τιμή 3 */
```

Παραστάσεις όπως η: i=i+3, όπου η μεταβλητή του αριστερού μέλους επαναλαμβάνεται στην αρχή του δεξιού μέλους, μπορούν να γραφούν στην πιο σύντομη μορφή: i+=3. Γενικά ισχύει ότι η φράση:

Παράσταση_1 τελεστής = Παράσταση 2

Ι

σοδυναμεί με την φράση:

Παράσταση_1 = (Παράσταση_2) Τελεστής (Παράσταση_1)

Οι παραπάνω παρενθέσεις δηλώνουν ***προτεραιότητα***. Για παράδειγμα, η φράση x*=y+1 ισοδυναμεί με την x=x*(y+1) **όχι** με την x=x*y+1.

1.5.2 ΣΧΕΣΙΑΚΟΙ ΚΑΙ ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ ΤΗΣ C.

Οι σχεσιακοί τελεστές συγκρίνουν δύο τιμές και επιστρέφουν την τιμή **true** ή **false** ανάλογα με το αποτέλεσμα της σύγκρισης. Οι λογικοί τελεστές συνενώνουν τέτοιες προτάσεις μεταξύ τους. Δηλαδή κάνουν πράξεις με τις τιμές **true / false**. Στους παρακάτω πίνακες φαίνονται οι συγκριτικοί και οι λογικοί τελεστές.

Τελεστής	**Ενέργεια**
>	Μεγαλύτερο από
>=	Μεγαλύτερο από ή ίσο με
<	Μικρότερο από
<=	Μικρότερο από ή ίσο με
==	Ίσο
!=	Άνισο (διάφορο)

Τελεστής	**Ενέργεια**
&&	AND
||	OR
!	NOT

Τα αποτελέσματα των λογικών τελεστών βρίσκονται βάση των **πινάκων αλήθειας** για τον κάθε τελεστή στον παρακάτω πίνακα φαίνονται οι **πινάκες αλήθειας** για τους τρεις τελεστές της C. Συμβολίζουμε με 0 την τιμή false και 1 την τιμή true.

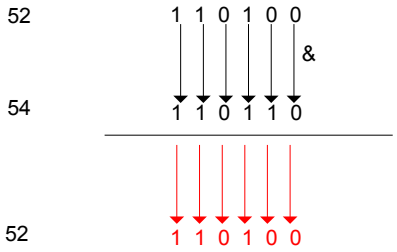
		Αποτελέσματα		
A	**B**	**A&&B**	**A||B**	**!A**
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Σε μία πρόταση μπορούμε να συνενώνουμε οποιονδήποτε αριθμό σχεσιακών τελεστών αρκεί, για να είναι σωστό το αποτέλεσμα, να τηρούνται οι προτεραιότητες του επόμενου πίνακα.

Βαθμίδα προτεραιότητας	Τελεστές
Υψηλότερη	!
↓	> >= < <=
	== !=
	&&
Χαμηλότερη	||

1.5.3 BITWISE ΤΕΛΕΣΤΕΣ.

Οι bitwise τελεστές είναι λογικοί τελεστές οι οποίοι ενεργούν σε κάθε bit της δυαδικής αναπαράσταση ενός αριθμού. Δηλαδή έστω ότι έχω δύο αριθμούς 52 και 54 αν κάνω το λογικό **ΚΑΙ (&&)** ανάμεσα σε αυτούς τους αριθμούς το αποτέλεσμά μου θα είναι 1 δηλαδή **true** αφού είναι και οι δύο μη μηδενική και η C αντιλαμβάνεται τις μη μηδενικές τιμές σαν true. Αν τώρα κάνω το λογικό **bitwise και** ανάμεσα στους δύο αριθμούς θα έχω σαν αποτέλεσμα τον αριθμό 52 όπως φαίνεται και στο παρακάτω σχήμα.



Οι bitwise τελεστές της C φαίνονται στον παρακάτω πίνακα

Τελεστής	Ενέργεια
&	Bitwise and
|	Bitwise or
^	Bitwise xor
~	Bitwise not
<<	Κύλιση προς τα αριστερά
>>	Κύλιση προς τα δεξιά

Οι τρεις τελεστές and, or και not έχουν τους ίδιους πίνακες αλήθειας όπως και πριν. Οι τελεστές κύλισης ουσιαστικά *μετακινούν την δυαδική λέξη είτε προς τα δεξιά είτα προς τα αριστερά τόσες θέσεις όσες περνούν σαν όρισμα*. Για παράδειγμα η έκφραση 52 >> 2 μετακινεί την ψηφιολέξη 110100 δύο θέσεις προς τα δεξιά και το αποτέλεσμα είναι 1101 δηλαδή 13 αντίστοιχα 52 << 2 δίνει 11010000 δηλαδή 208. ***Παρατηρείστε ότι η κύλιση προς τα αριστερά ισοδυναμεί με πολλαπλασιασμό του αριθμού με την αντίστοιχη δύναμη του 2 (2 θέσεις δηλαδή 2^2=4 και 4*52=208) ενώ προς τα δεξιά με διαίρεση (2 θέσεις δηλαδή 2^2=4 και 52/4=13).*** Ο τρόπος αυτός για πολλαπλασιασμό

ακεραίων με δυνάμεις του 2 συνήθως προτιμάται γιατί έχει καλύτερη χρονική πολυπλοκότητα.

Τέλος ο πίνακας αλήθειας του xor δίνεται παρακάτω.

Έντελα		**Αποτελέσματα**
A	**B**	**XOR**
0	0	0
0	1	1
1	0	1
1	1	0

1.5.4 TYPE CASTING

Το **casting** είναι ένα από τα ισχυρότερα εργαλεία της C γιατί μας επιτρέπει την χρήση των ίδιων μεταβλητών με διαφορετικούς περιστασιακούς τύπους. Για παράδειγμα ένας **float** δεν μπορεί να συμμετέχει σε μια πράξη του τελεστή % έτσι μπορούμε να κάνουμε **type cast** στην μεταβλητή **float** και η C θα μετατρέψει την μεταβλητή αυτή σε **int** για παράδειγμα. Το **type casting** γίνεται βάση της γενικής μορφής:

(τύπος) τιμή;

Στο παρακάτω πρόγραμμα φαίνεται ένα **type casting** από **float** σε **int**.

```
#include <stdio.h>

int main(){
        float f;

        f = 1.5;
        /*Η f εμφανίζεται σαν ακέραιος*/
        printf(" %d ",(int) f);
        return 0;
}
```

2. ΒΑΣΙΚΗ ΕΙΣΟΔΟΣ – ΕΞΟΔΟΣ

Οι βασικές εντολές εισόδου και εξόδου στην C είναι η printf και η scanf. Οι δύο αυτές συναρτήσεις είναι μέλη της βασικής βιβλιοθήκης της C και για να χρησιμοποιηθούν πρέπει να στην αρχή κάθε προγράμματός να περικλείουμε το αρχείο stdio.h. Στο κεφάλαιο αυτό θα εξετάσουμε αναλυτικά τις συναρτήσεις.

2.1 Η printf()

Η συνάρτηση **printf()** εκτυπώνει αλφαριθμητικό στην οθόνη και έχει το ακόλουθο πρωτότυπο:

*int printf(char *αλφαριθμητικό-ελέγχου, ...);*

Τα αποσιωπητικά υποδηλώνουν μια λίστα ορισμάτων με μεταβαλλόμενο μέγεθος. Η συνάρτηση **printf()** επιστρέφει τον αριθμό των χαρακτήρων που εξάγει. Εάν συμβεί ένα σφάλμα, επιστρέφει έναν αρνητικό αριθμό. Ελάχιστοι προγραμματιστές ασχολούνται με την επιστρεφόμενη τιμή της **printf()**.

Το αλφαριθμητικό ελέγχου μπορεί να περιέχει δύο τύπους στοιχείων: χαρακτήρες για έξοδο και κωδικούς μορφοποίησης. Όλοι οι κωδικοί μορφοποίησης ξεκινούν με το σύμβολο %. Ένας τέτοιος κωδικός καθορίζει τον τρόπο εμφάνισης του ορίσματος στο οποίο αντιστοιχεί. Οι κωδικοί μορφοποίησης και τα ορίσματα τους αντιστοιχίζονται από τα αριστερά προς τα δεξιά και πρέπει να υπάρχουν τόσα ορίσματα όσοι είναι και οι κωδικοί μορφοποίησης.

Οι κωδικοί μορφοποίησης που δέχεται η **printf()** φαίνονται στον παρακάτω πίνακα:

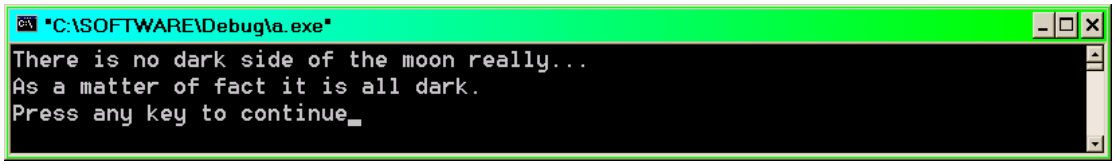
Κωδικός	**Μορφοποίηση**
%c	Χαρακτήρας
%d	Προσημασμένοι δεκαδικοί ακέραιοι
%i	Προσημασμένοι δεκαδικοί ακέραιοι
%e	Εκθετική αναγραφή(με πεζο e)
%E	Εκθετική αναγραφή
%f	Δεκαδικοί αριθμοί κινητής υποδιαστολής
%g	Χρησιμοποιεί τον %e ή τον %f – οτιδήποτε είναι μικρότερο
%G	Χρησιμοποιεί τον %E ή τον %f – οτιδήποτε είναι μικρότερο
%o	Μη-προσημασμένος οκταδικός
%s	Αλφαριθμητικό χαρακτήρων
%u	Μη-προσημασμένος δεκαδικός
%x	Μη-προσημασμένος δεκαεξαδικός (πεζα γράμματα)
%X	Μη-προσημασμένος δεκαεξαδικός

	(κεφαλαία γράμματα)
%p	Εμφάνιση δείκτη
%n	Το σχετιζόμενο όρισμα είναι ένας δείκτης προς έναν ακέραιο στον οποίο τοποθετείται το πλήθος των χαρακτήρων που έχουν γραφτεί μέχρι τώρα
%%	Εκτυπώνει το σύμβολο %

Παράδειγμα 1 (εκτύπωση απλών αλφαριθμητικών):

```
#include <stdio.h>
int main()
{
    printf("There is no ");
    printf("dark side of the moon really");
    printf("...\n");
    printf("As a matter of fact ");
    printf("it is all dark.");
    printf("\n");
    return 0;
}
```

Η εκτέλεση του προγράμματος φαίνεται στο ακόλουθο σχήμα:



Παρατηρούμε ότι αν στην έξοδο δεν έχουμε αλλαγή γραμμής αν γράψουμε διαφορετικές κλήσεις της printf(). Για να αλλάξει η γραμμή πρέπει να γράψουμε τον χαρακτήρα αλλαγής γραμμής (\n);

Παράδειγμα2 (εκτύπωση τιμής μεταβλητής):

```
float x = 3.2;
printf("O arithos x einai isos me %f.\n",x);
```

Η εκτέλεση του προγράμματος φαίνεται στο ακόλουθο σχήμα:

Παρατηρούμε ότι ο αριθμός εμφανίστηκε με ακρίβεια 6 δεκαδικών ψηφίων. Αν θέλουμε να ορίσουμε το πλήθος των δεκαδικών ψηφίων (π.χ. σε 3) πρέπει να γράψουμε:

```
printf("O arithos x einai isos me %.3f.\n",x);
```

Επίσης, αν θέλουμε να ορίσουμε συνολικό πλήθος των ψηφίων ενός αριθμούς (δεκαδικού ή ακεραίου) μπορούμε να βάλουμε τον αντίστοιχο αριθμό μπροστά από την τελεία. Για παράδειγμα:
*printf("Ο arithos x einai isos me **%6.3f**.\n",x);*

Τέλος, η εντολή:
*printf("Ο arithos x einai isos me **%.3f**.\n",x);*

αν το x είναι ακέραιος, εκτυπώνει μηδενικά πριν από τον αριθμό (για παράδειγμα αν x=2) τότε θα εκτυπωθεί 002.

2.2 Η scanf()

Η συνάρτηση αυτή έχει ως σκοπό την είσοδο δεδομένων από το πληκτρολόγιο. Το πρωτότυπο της **scanf()** είναι:

*int scanf(char *αλφαριθμητικό-ελέγχου, ...);*

Το αλφαριθμητικό ελέγχου αποτελείται κυρίως από κωδικούς μορφοποίησης. Ωστόσο, μπορεί να περιέχει και άλλους χαρακτήρες. Οι κωδικοί μορφοποίησης καθορίζουν τον τρόπο με τον οποίο διαβάζει η **scanf()** τις πληροφορίες τις οποίες και τοποθετεί στις μεταβλητές που **δείχνουν** τα ορίσματα μετά από το αλφαριθμητικό ελέγχου. Όπως και με την **printf()** έτσι και εδώ οι κωδικοί αντιστοιχίζονται απο τα αριστερά προς τα δεξιά και πρέπει να υπάρχουν όσοι κωδικοί τόσα ορίσματα. Στον παρακάτω πίνακα φαίνονται οι κωδικοί μορφοποίησης της **scanf()**.

Κωδικός	**Μορφοποίηση**
%c	Ανάγνωση χαρακτήρα
%d	Ανάγνωση δεκαδικού ακέραιου
%i	Ανάγνωση δεκαδικού ακέραιου
%e	Ανάγνωση αριθμού κινητής υποδιαστολής
%f	Ανάγνωση αριθμού κινητής υποδιαστολής
%g	Ανάγνωση αριθμού κινητής υποδιαστολής
%o	Ανάγνωση οκταδικού
%s	Ανάγνωση Αλφαριθμητικού
%u	Ανάγνωση μη-προσημασμένου δεκαδικού
%x	Ανάγνωση δεκαεξαδικού
%p	Ανάγνωση δείκτη
%n	Λαμβάνει έναν ακέραιο ίσο με τον αριθμό των χαρακτήρων που έχουν διαβαστεί μέχρι τώρα
%[]	Έλεγχος για ένα σύνολο χαρακτήρων

Η διαφορά με την printf είναι ότι οι μεταβλητές που περνάμε σαν ορίσματα πρέπει να αλλάξουν τιμές. Για τον λόγο αυτό, όπως θα δούμε σε επόμενο κεφάλαιο, πρέπει να δίνουμε την διεύθυνση μίας μεταβλητής (σύμβολο &). Για παράδειγμα:

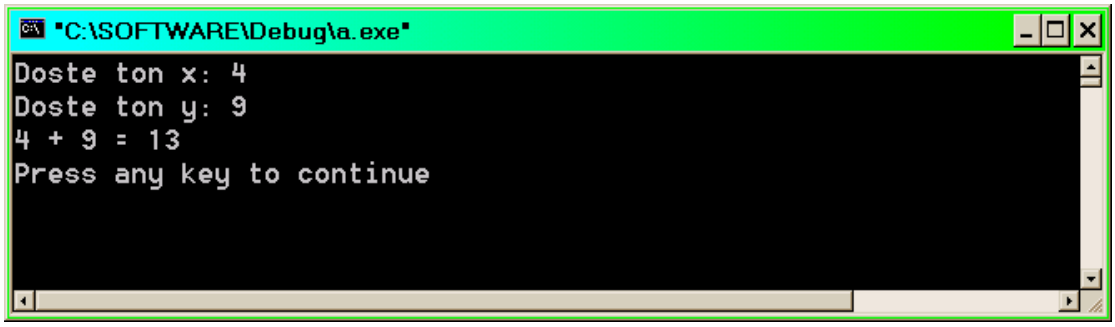
```
int a;
scanf(“%d”,&a);
```

Παράδειγμα:

Στο ακόλουθο πρόγραμμα ζητείται από τον χρήστη να δώσει 2 ακεραίους (x και y), υπολογίζεται το άθροισμά τους (apotelesma) και στην συνέχεια εκτυπώνεται:

```
#include <stdio.h>
int main()
{
    int x,y,apotelesma;
    printf("Doste ton x: ");
    scanf("%d",&x);
    printf("Doste ton y: ");
    scanf("%d",&y);
    apotelesma = x + y;
    printf("%d + %d = %d\n",x,y,apotelesma);
    return 0;
}
```

Στην ακόλουθη εικόνα φαίνεται ένα παράδειγμα εκτέλεσης του προγράμματος:



3. ΕΝΤΟΛΕΣ ΕΛΕΓΧΟΥ ΚΑΙ ΕΠΑΝΑΛΗΨΗΣ

3.1 ΟΙ ΕΝΤΟΛΕΣ ΕΛΕΓΧΟΥ ΡΟΗΣ

3.1.1 Η ΕΝΤΟΛΗ IF-ELSE

Η εντολή **if** είναι μία από τις εντολές επιλογής της γλώσσας C. Στην ποιο απλή της μορφή την συναντάμε με τον γενικό τύπο:

if(συνθήκη) εντολή;

Μία τέτοια γραμμή κώδικα θα επιτρέψει την εκτέλεση της εντολής μόνο αν η συνθήκη είναι αληθής. Συνθήκη μπορεί να είναι οποιαδήποτε έγκυρη έκφραση της C. Ένα παράδειγμα χρήσης της if δίνεται παρακάτω.

if(10 > 9) printf(“true”);

Πρέπει να σημειωθεί ότι σε αυτή την μορφή η **if** συναντάται πολύ σπάνια. Η πιο συνηθισμένη της μορφή είναι αυτή που περιέχει ένα τμήμα κώδικα μετά την συνθήκη.

```
if(συνθήκη){
    εντολή
    εντολή
        .
        .
    εντολή
    }
```

Όπως φαίνεται μια τέτοιας μορφή εντολή θα επιτρέψει την εκτέλεση του μπλοκ των εντολών μόνο αν η συνθήκη είναι αληθής. Κατά αντιστοιχία συνθήκη μπορεί να είναι οποιαδήποτε έγκυρη έκφραση της C. Ένα παράδειγμα χρήσης τέτοιας μεταβλητής φαίνεται παρακάτω.

```
 if(num > 0){
    printf(“This is”);
    printf(“an example”);
     printf(“of block if ”);
}
```

Πολύ συχνά βέβαια έχουμε την ανάγκη να εκτελέσουμε εντολές όταν δεν ισχύει η συνθήκη της **if**. Κάτι τέτοιο θα μπορούσαμε να το πετύχουμε με μία δεύτερη **if** στην οποία θα είχαμε σαν συνθήκη την άρνηση της συνθήκης της πρώτης **if**. Η C μας δίνη την δυνατότητα να συνδυάσουμε την **if** με την **else** η οποία εκτελείτε αν δεν ισχύει η συνθήκη της **if**. Η γενική μορφή μιας τέτοιας εντολή είναι:

```
if(συνθήκη){
    εντολή
    εντολή
```

```
        .
        .
    εντολή
}else{
    εντολή
    εντολή
        .
        .
    εντολή
}
```

Όμοια με παραπάνω δίνεται παράδειγμα χρήσης της εντολή **if-else**:

```
 if(num > 0){
    printf(“num is ”);
    printf(“bigger than”);
    printf(“ 0”);
}else{
    printf(“num is ”);
    printf(“smaller than”);
    printf(“ 0”);
                }
```

Η C διαθέτει επίσης έναν «τριαδικό» τελεστή: το **?**. Ένας τριαδικός τελεστής απαιτεί τρεις τελεσταίους. Ο τελεστής **?** χρησιμοποιείται σαν υποκατάστατο εντολών του τύπου:

```
if(συνθήκη) μεταβλητή = εκφραση1;
else μεταβλητή = έκφραση2;
```

Η γενική του μορφή είναι:

```
μεταβλητή = συνθήκη ? έκφραση1 : έκφραση2;
```

Παράδειγμα:
max = (x>y)?x:y;

(το max παίρνει την μέγιστη τιμή ανάμεσα στον x και y).

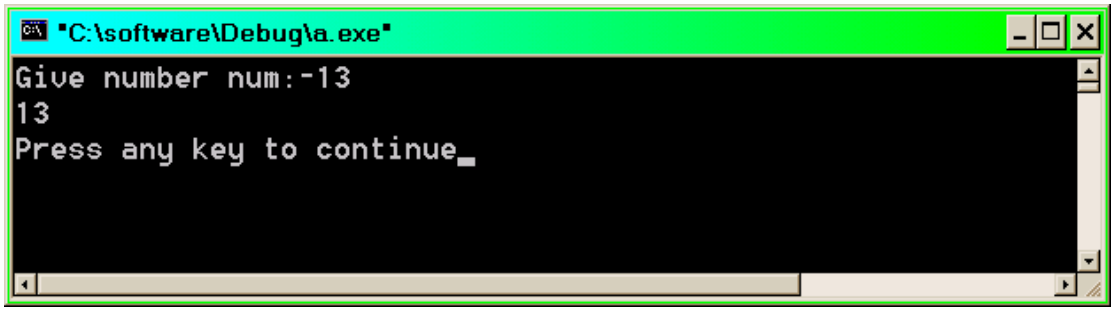
Εδώ, η συνθήκη είναι μια έκφραση η οποία αποτιμάται σε **true** ή **false**. Εάν είναι **true**, στην μεταβλητή εκχωρείται η τιμή της *εκφραση1* αλλιώς αυτή της *έκφραση2*. Ο λόγος ύπαρξης του τελεστή **?** είναι ότι ένας μεταγλωττιστής C μπορεί να παράγει πολύ αποτελεσματικό κώδικα όταν χρησιμοποιεί αυτόν τον τελεστή, αντί της ισοδύναμης **if/else**.

Σημείωση

Πρέπει να προσέχουμε ότι όταν θέλουμε να ελέγξουμε ισότητα ανάμεσα σε δύο αριθμούς δεν βάζουμε =, αλλά το λογικό ==, διαφορετικά δεν γίνεται έλεγχος, αλλά ανάθεση τιμής. Αυτό ισχύει, όχι μόνο για το if-else αλλά για όλες τις εντολές ελέγχου που περιέχουν συνθήκες.

Παραδειγμα

```
#include <stdio.h>

main()
{
int num;
int apo;
printf("Give number num:");
scanf("%d",&num);

if (num>0)
apo=num;
else
apo=-num;
printf("%d\n",apo);

}
```



3.1.2 Η ΕΝΤΟΛΗ SWITCH

Αν και η **if** είναι πολύ καλή για την επιλογή μεταξύ δύο εναλλακτικών καταστάσεων, πολύ σύντομα μπορεί να καταντήσει κουραστική εάν αντιμετωπίζεται *περισσότερες από δύο εναλλακτικές επιλογές*. Η λύση που παρέχει η C σ'αυτό το πρόβλημα είναι η εντολή **switch**. Η εντολή αυτή είναι πολλαπλών επιλογών και χρησιμοποιείται για την επιλογή μιας, από αρκετές εναλλακτικές διαδρομές της ροής εκτέλεσης του προγράμματος. Ο τρόπος που λειτουργεί η **switch** είναι ο εξής: Μία τιμή ελέγχεται διαδοχικά έναντι μιας λίστας σταθερών (ακεραίων ή χαρακτήρων). Όταν βρεθεί ένα «ταίριασμα», εκτελείται η αλληλουχία εντολών που σχετίζεται μ'αυτό το «ταίριασμα»(ή μ'αυτη την περίπτωση **case**). Η γενική μορφή της **switch** είναι:

```
switch(τιμή){
case σταθερά1:
      αλληλουχία εντολών;
      break;
case σταθερά2:
      αλληλουχία εντολών;
      break;
          .
          .
```

.

```
default:
    αλληλουχία εντολών;
    break;
```

Η αλληλουχία των εντολών της **default** εκτελείται σε περίπτωση που δεν βρεθεί ούτε ένα ταίριασμα με τις προηγούμενες **case**. Η χρήση της **default** είναι προαιρετική για την σύνταξη της εντολής **switch**.

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

        printf(" Enter a number between 1 and 4 ");
        scanf("%d",&i);
        switch(i){
                case 1: printf("one");
                        break;
                case 2: printf("tow");
                        break;
                case 3: printf("three");
                        break;
                case 4: printf("four");
                        break;
                default: printf("unrecognized Number");
        }
        return 0;
}
```

3.1.3 ΟΙ ΕΝΤΟΛΕΣ BREAK – CONTINUE – GOTO

Οι εντολές αυτές υπάρχουν στην C αν και η χρήση τους δεν συνιστάται γιατί πρόκειται για εντολές αυθαίρετου ελέγχου ροής. Αν και η **break** είναι αναγκαία για την **switch** η χρήση της περιορίζεται σε αυτήν και μόνο την διαδικασία. Ο ορισμός της **break** είναι η άνευ όρων διακοπή των εντολών του τμήματος στο οποίο βρίσκεται και στην **switch** κάτι τέτοιο μας είναι χρήσιμο για την εγκατάλειψη του τμήματος **switch** μετά από ένα σωστό ταίριασμα. Γενικά όμως δεν συνιστάται η χρήση της.

Η εντολή **continue** βρίσκεται σε τμήματα κώδικα επαναλήψεων και στέλνει την ροή του προγράμματος στην συνθήκη ελέγχου της επανάληψης. Η χρήση της είναι σπάνια όχι μόνο επειδή είναι κακός τρόπος προγραμματισμού αλλά γιατί είναι και πολύ δύσκολο να υπάρξει μια κατάσταση στην οποία να χρειαστεί μια τέτοια εντολή.

Η εντολή **goto**. Πρόκειται για μια εντολή που ήταν ο κυρίως λόγος για την ανακάλυψη του δομημένου προγραμματισμού. Αν και είναι από τις αρχαιότερες εντολές εν ενεργεία η χρήση της πρέπει να αποφεύγεται συστηματικά. Από το 1970 και μετά η χρήση της **goto** συστηματικά ελαττώνεται και σήμερα σχεδόν όλα τα προγράμματα δεν κάνουν χρήση αυτής της εντολής. Σε όλη τη διάρκεια του εξαμήνου δεν θα ξανά ασχοληθούμε με την εντολή **goto** (και πιθανότατα ποτέ...).

3.2 ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ (LOOPs)

3.2.1 Η ΕΝΤΟΛΗ WHILE

Μια εντολή που διαθέτει η C για την δημιουργία βρόχων (loops) είναι η **while**. Η **while** ακολουθεί την γενική μορφή:

```
while(συνθήκη){
    εντολή;
    εντολή;
        .
        .
    εντολή;
}
```

Ο βρόχος **while** εκτελεί τις εντολές εντός του τμήματος κώδικα όσο η συνθήκη είναι αληθής. Εξυπακούεται ότι αν η συνθήκη είναι εξ'αρχής ψευδής το μπλοκ των εντολών δεν εκτελείται.

Στο παρακάτω πρόγραμμα φαίνεται η τυπική χρήση ενός βρόχου **while**:

```
#include <stdio.h>

int main(){
        int i=0;

        while(i<10){
                printf("%d",i);
                i=i+1;
        }
        return 0;
}
```

3.2.2 Η ΕΝΤΟΛΗ FOR.

Η εντολή **for** επιτρέπει την επαναλαμβανόμενη εκτέλεση μίας ή περισσοτέρων εντολών και θεωρείται σαν η πιο ευέλικτη μορφή βρόχου της γλώσσας C. Η γενική μορφή για την επανάληψη ενός τμήματος κώδικα είναι:

```
for(αρχικοποίηση;συνθήκη-ελέγχου;μεταβολή μετρητή){
        εντολές1....N
}
```

Το τμήμα ***αρχικοποίησης*** χρησιμοποιείται για την παροχή μιας αρχικής τιμής στον μετρητή του βρόχου και εκτελείται μόνο μία φορά κατά την εισαγωγή στο βρόχο. Το δεύτερο τμήμα ***συνθήκης-ελέγχου*** ελέγχει τον μετρητή έναντι μίας τιμής προορισμού. Τέλος το τμήμα ***μεταβολής-μετρητή*** μεταβάλει την τιμή του μετρητή έτσι ώστε να φτάσει ο μετρητής στην τιμή προορισμού.

Ένα απλό παράδειγμα εφαρμογής της **for** φαίνεται στο ακόλουθο πρόγραμμα:

```
#include <stdio.h>

int main(){
        int i=0;

        for(i=1;i<=10;i++){
                printf("%d ",i);
        }
        return 0;
}
```

Το παραπάνω πρόγραμμα προφανώς, εκτυπώνει όλους τους ακεραίους από 1 ως 10. Η εντολή **for** έχει χαρακτηριστεί σαν η πιο ευέλικτη γιατί ουσιαστικά τα τρία τμήματα που παρουσιάσαμε παραπάνω μπορούν να χρησιμοποιηθούν πολύ διαφορετικά έτσι ώστε να προσφέρουν μεγάλη διευκόλυνση στον προγραμματιστή.

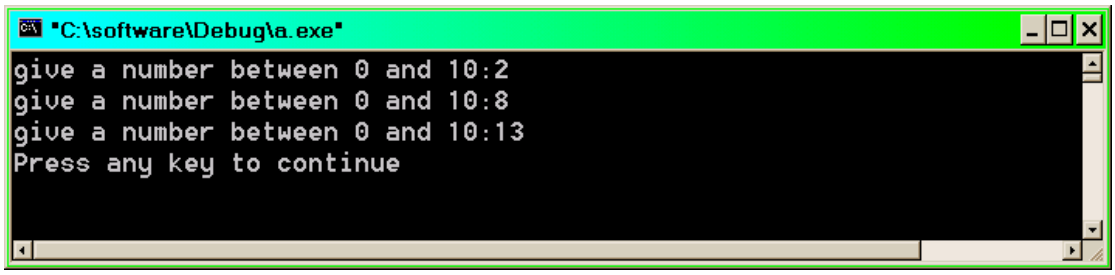
3.2.3 Η ΕΝΤΟΛΗ DO

Η τελευταία εντολή βρόχου της C είναι ο βρόχος **do**, με την ακόλουθη γενική μορφή:

```
do{
    εντολές;
}while(συνθήκη)
```

Εάν ο βρόχος επαναλαμβάνει μόνο μία εντολή, τα άγκιστρα δεν είναι υποχρεωτικά αλλά συνιστάται η χρήση τους για την καλύτερη αναγνωσιμότητα του κώδικα. Ο βρόχος **do** επαναλαμβάνει την/τις εντολή/ές του μπλοκ του όσο η συνθήκη είναι αληθής και σταματά τη στιγμή που η συνθήκη γίνεται ψευδής. Το πλεονέκτημα ενός βρόχου **do** είναι το γεγονός ότι οι εντολές του μπλοκ του εκτελούνται ***τουλάχιστον μία φορά*** δεδομένου ότι η συνθήκη βρίσκεται στο τέλος του βρόχου. Το παρακάτω πρόγραμμα δείχνει μια τυπική εφαρμογή του βρόχου **do**.

```
#include <stdio.h>

int main(){
	int i=0;

	do{
	printf(“give a number between 0 and 10”);
	scanf(“%d”,&i);
	}while((i>=0)&&(i<=10));
	return 0;
}
```



Στο πρόγραμμα αυτό ο χρήστης δίνει αριθμούς μέχρι να δώσει έναν έγκυρο αριθμό δηλαδή μεταξύ 0 και 10.

4. ΠΙΝΑΚΕΣ

4.1 ΧΡΗΣΙΜΟΤΗΤΑ ΠΙΝΑΚΩΝ

Ας υποθέσουμε ότι θέλουμε να υπολογίσουμε τον μέσο όρο 100 αριθμών, τους οποίους διαβάζουμε, είτε από το πληκτρολόγιο (με την scanf()), είτε (όπως θα δούμε αργότερα) από αρχείο. Με όσα γνωρίζουμε ως τώρα θα έπρεπε να ορίσουμε 100 μεταβλητές! Η χρήση των πινάκων δίνει λύση στο πρόβλημα αυτό. Συγκεκριμένα, οι πίνακες είναι ένας τρόπος ομαδοποίησης πολλών μεταβλητών του ίδιου τύπου. Στο ακόλουθο σχήμα περιγράφεται ένας πίνακας μίας διάστασης 100 θέσεων, οποίος περιέχει ακεραίους:

Περιεχόμενο στοιχείου	21	31	1	-4		4	2	1
Θέση στοιχείου	0	1	2	3	.....	97	98	99

Όπως φαίνεται και στο σχήμα, στην C θεωρούμε ότι οι θέσεις των στοιχείων μετρούνται από το 0 (όχι από το 1).

4.2 ΟΡΙΣΜΟΣ ΠΙΝΑΚΩΝ ΣΤΗΝ C

Για να ορίσουμε έναν πίνακα ***μίας διάστασης*** στην C (αλλιώς το ονομάζουμε array) γράφουμε το εξής:
ΤΥΠΟΣ ΟΝΟΜΑ_ΠΙΝΑΚΑ[ΜΕΓΕΘΟΣ];

Για παράδειγμα η εντολή:
float x[100];

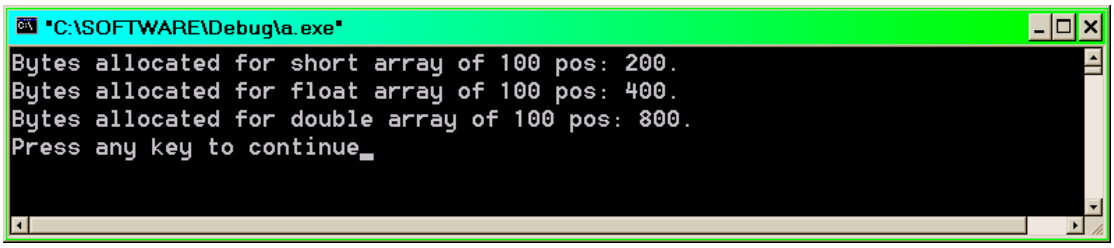
ορίζει ένα array ακεραίων 100 θέσεων με όνομα x.

Όταν δηλώνουμε ένα array ο compiler της C δεσμεύει όση μνήμη χρειάζεται για να κρατήσει όλα τα στοιχεία του array αυτού. Στο παραπάνω παράδειγμα (αν θεωρήσουμε ότι στα περισσότερα συστήματα ο float είναι 4 bytes) το συνολικό μέγεθος της δεσμευόμενης μνήμης είναι 4*100 = 400 bytes. Στο ακόλουθο παράδειγμα, υπολογίζονται τα bytes της μνήμης που δεσμεύονται για 3 διαφορετικούς τύπους arrays:

```
#include <stdio.h>

int main()
{
        short x[100];
        float a[100];
        double d[100];
        printf("Bytes allocated for short array of 100 pos: %d.\n",sizeof(x));
        printf("Bytes allocated for float array of 100 pos: %d.\n",sizeof(a));
```

```
        printf("Bytes allocated for double array of 100 pos: %d.\n",sizeof(d));
        return 0;
}
```

Το αποτέλεσμα της εκτέλεσης του παραπάνω προγράμματος φαίνεται στην ακόλουθη εικόνα:



Σημείωση

Η συνάρτηση sizeof() επιστρέφει το μέγεθος (σε bytes) μίας μεταβλητής, ενός array, ή όπως θα δούμε αργότερα μίας δομής που εμείς έχουμε κατασκευάσει. Για τις προκαθορισμένες δομές (int, short κτλ) ΔΕΝ υπάρχουν standards, αλλά το μέγεθός τους μπορεί να εξαρτηθεί από το εκάστοτε λειτουργικό σύστημα. Ωστόσο, ο short έχει πάντα μικρότερο ή ίσο μέγεθος από τον int, ο double πάντα μεγαλύτερο ή ίσο μέγεθος από τον float κλτ. Συνήθως

Για να ορίσουμε έναν ***διδιάστατο*** πίνακα στην C γράφουμε:
ΤΥΠΟΣ ΟΝΟΜΑ_ΠΙΝΑΚΑ[ΠΛΗΘΟΣ_ΓΡΑΜΜΩΝ][ΠΛΗΘΟΣ_ΣΤΗΛΩΝ];

Για παράδειγμα, η δήλωση:
double x[100][20];

ορίζει έναν 2-D πίνακα 100 γραμμών και 20 στηλών, δηλαδή συνολικά 100*20 = 2000 στοιχείων. Το συνολικό μέγεθος του πίνακα σε bytes είναι 2000*8 = 16000 bytes.

4.3 ΑΡΧΙΚΟΠΟΙΗΣΗ ΠΙΝΑΚΩΝ

Για να αρχικοποιήσουμε έναν πίνακα (π.χ. ακεραίων) κάνουμε το εξής:
int array[5] = {2,3,5,1,5};

Η παραπάνω δήλωση σημαίνει ότι το $1^{ο}$ στοιχείο έχει τιμή 2, το $2^{ο}$ τιμή 3, το $3^{ο}$ τιμή 5 κτλ.. Αν θέλουμε να αρχικοποιήσουμε έναν πίνακα χαρακτήρων (string) τότε μπορούμε να χρησιμοποιήσουμε τα διπλά εισαγωγικά:

char name[10] = "ABCDE";
ή

char name[] = "ABCDE";

Στην 1η περίπτωση (δηλώνουμε αρχικό μέγεθος πίνακα ίσο με 10), ο compiler θα δεσμεύσει 10 bytes, θα αρχικοποιήσει τα πρώτα 5 bytes (χαρακτήρες ABCDE) και τα υπόλοιπα θα είναι ίσα με NULL. Στην 2η περίπτωση, ο compiler θα δεσμεύσει αυτόματα όσα bytes χρειάζονται για την αρχικοποίηση, δηλαδή 5. Αυτό ισχύει για όλους τους τύπους δεικτών. Για παράδειγμα ο ακόλουθος κώδικας:

int A[] = {1,2,3,4,5,6};
printf("%d\n",sizeof(A));

θα τυπώσει 26, που σημαίνει ότι έχου δεσμευτεί 26 bytes, άρα το μέγεθος του πίνακα γίνεται ίσο με 26/4 = 6.

Αν θέλουμε να αρχικοποιήσουμε έναν διδιάστατο πίνακα, κάνουμε το ίδιο. Για παράδειγμα η δήλωση:
int a[2][2] = {1,2,3,4};

δημιουργεί τον πίνακα:

1	2
3	4

4.4 ΠΡΟΣΠΕΛΑΣΗ ΣΤΟΙΧΕΙΩΝ ΠΙΝΑΚΩΝ

Για να προσπελάσουμε ένα στοιχείο ενός array γράφουμε:
ΟΝΟΜΑ_ΠΙΝΑΚΑ[ΘΕΣΗ_ΣΤΟΙΧΕΙΟΥ]

Αν ο πίνακας είναι διδιάστατος, τότε για να προσπελάσουμε ένα στοιχείο γράφουμε:
ΟΝΟΜΑ_ΠΙΝΑΚΑ[ΓΡΑΜΜΗ_ΣΤΟΙΧΕΙΟΥ][ΣΤΗΛΗ_ΣΤΟΙΧΕΙΟΥ]

Στο ακόλουθο παράδειγμα, αρχικοποιούμε τα στοιχεία ενός πίνακα ακεραίων Α (όπως είδαμε στην προηγούμενη παράγραφο) και στην συνέχεια τα εκτυπώνουμε):
int x[] = {10,20,30};
printf("To prwto stoixeio einai iso me: %d.\n",x[0]);
printf("To deytero stoixeio einai iso me: %d.\n",x[1]);
printf("To trito stoixeio einai iso me: %d.\n",x[2]);

Πρέπει να προσέξουμε ότι η αρίθμηση των θέσεων του πίνακα ξεκινάει από το 0, ***όχι*** από το 1. Το ίδιο ισχύει και για τους 2-D πίνακες: η αρίθμηση των γραμμών και των στηλών ξεκινάει από το 0.

Στο ακόλουθο παράδειγμα, ορίζουμε έναν δισδιάστατο πίνακα, τον αρχικοποιούμε, και εκτυπώνουμε τα στοιχεία του:
int x[2][2] = {1,2,3,4};
printf("stoixeio (0,0): %d.\n",x[0][0]);

```
printf("stoixeio (0,1): %d.\n",x[0][1]);
printf("stoixeio (1,0): %d.\n",x[1][0]);
printf("stoixeio (1,1): %d.\n",x[1][1]);
```

Συνήθως τα arrays χρησιμοποιούνται σε συνδυασμό με εντολές βρόγχων (loops). Στο ακόλουθο παράδειγμα, ορίζεται ένας πίνακας μίας διάστασης 100 θέσεων. Στην συνέχεια, ο πίνακας με χρήση ενός for loop παίρνει την μορφή:

0	2	.....	196	198

```
#include <stdio.h>
int main()
{
        int array[100];
        int i;
        for (i=0;i<100;i++)
                array[i] = 2*i;
        return 0;
}
```

Στο παραπάνω παράδειγμα, αν θέλαμε να αλλάξουμε το πλήθος των στοιχείων (π.χ. από 100 να το κάνουμε 50), θα έπρεπε να αλλάζαμε την δήλωση του πίνακα, αλλά και το for loop. Σε μεγαλύτερα προγράμματα αυτό θα ήταν πολύ χρονοβόρο. Ένας εναλλακτικός τρόπος είναι η χρήση σταθερών στην περιοχή του προεπεξεργαστή. Αυτό γίνεται με την εντολή **#define**. Η εντολή αυτή, δεν ορίζει μεταβλητές, ούτε σταθερές μεταβλητές, όπως η const, αλλά στην ουσία αντικαθιστά ένα όνομα με ένα άλλο. Για παράδειγμα, αν γράψουμε:

```
#define SIZE 10
```

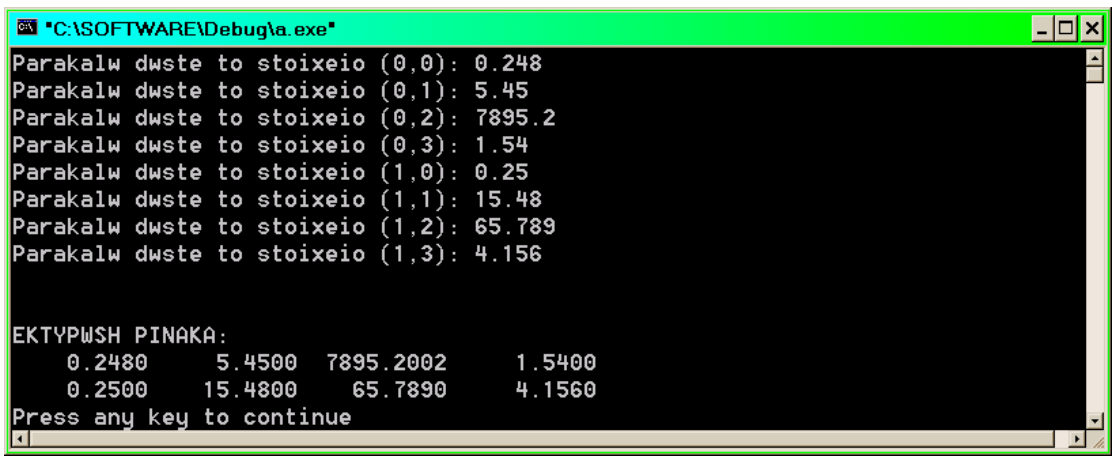
τότε, ο compiler όπου βρίσκει μέσα στον κώδικα την λέξη SIZE, θα την αντικαθιστά με την λέξη 10.

Στο πρόγραμμα που ακολουθεί:

- ορίζεται ένας δισδιάστατος πίνακας του οποίου οι διαστάσεις είναι σταθερές (ορίζονται με την #define)
- τα στοιχεία του πίνακα λαμβάνουν τιμές από την είσοδο (πληκτρολόγιο).
- τα στοιχεία τυπώνονται με μορφή του πίνακα που ξέρουμε (ανά γραμμή).

```
#include <stdio.h>

#define M 2 /* plithos grammwn */
#define N 4 /* plithos stilwn */

int main()
{
	float pinakas[M][N];  /* orismos pinaka */
	int i,j; /* metablites gia indexing*/

	/* LOOP 1: Diavasma timwn apo to pliktrologio (scanf)*/
	for (i=0;i<M;i++)
	{
		for (j=0;j<N;j++)
		{
			printf("Parakalw dwste to stoixeio (%d,%d): ",i,j);
			scanf("%f",&pinakas[i][j]);
		}
	}

	printf("\n\nEKTYPWSH PINAKA:\n");
	/* LOOP 2: Ektypwsi stoixeiwn pinaka (printf)*/
	for (i=0;i<M;i++)
	{
		for (j=0;j<N;j++)
		{
			printf("%10.4f ",pinakas[i][j]);
			/* ta stoixeia grafontai me akribeia 4 dekadikwn psifiwn */
			/* kai xrisimopoieitai synolika xwros 10 psifiwn */
		}
		printf("\n"); /* allagi gramis !*/
	}

	return 0;
}
```

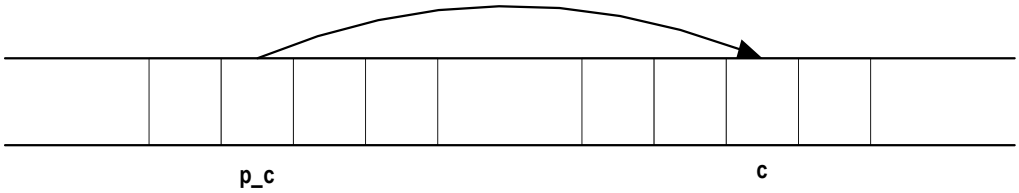
Ακολουθεί ένα παράδειγμα εκτέλεσης:



6. ΔΕΙΚΤΕΣ

6.1 ΤΙ ΕΙΝΑΙ ΟΙ ΔΕΙΚΤΕΣ

Οι δείκτες (pointers) είναι ίσως το πιο χρήσιμο εργαλείο της γλώσσας C. Στην ουσία ένας δείκτης είναι μία ακέραια μεταβλητή η οποία περιέχει την διεύθυνση μίας μεταβλητής. Στην επόμενη εικόνα φαίνεται ο τρόπος με τον οποίο είναι οργανωμένη η μνήμη του υπολογιστή:



Η μνήμη αποτελείται από μία σειρά κελιών (bytes) τα οποία έχουν διαδοχική αρίθμηση (διευθύνσεις). Όταν δηλώνουμε μία μεταβλητή, (π.χ. int a), ο compiler δεσμεύει τόσο χώρο μνήμης, όσο χρειάζεται η μεταβλητή αυτή. Για την περίπτωση του int δεσμεύονται 4 κελιά (δηλαδή 4 bytes). Στο παραπάνω σχήμα λοιπό, θεωρούμε ότι c είναι μία μεταβλητή χαρακτήρα και ότι p_c είναι ένας δείκτης που δείχνει σε αυτήν.

6.2 ΔΗΛΩΣΗ ΚΑΙ ΧΡΗΣΗ ΔΕΙΚΤΩΝ

Οι δείκτες δηλώνονται ως εξής:
*ΤΥΠΟΣ *ΟΝΟΜΑ;*

Για παράδειγμα, η δήλωση:
*int *ptr;*

δηλώνει έναν δείκτη με όνομα ptr, οποίος θα χρησιμοποιηθεί για να δείχνει σε ακέραιο.

Υπάρχουν δύο πολύ σημαντικοί τελεστές που θα τους χρησιμοποιούμε όταν δουλεύουμε με δείκτες:

1. Ο τελεστής ***** ονομάζεται τελεστής έμμεσης αναφοράς, και όταν εφαρμόζεται σε έναν δείκτη προσπελάζει **το περιεχόμενο του δείκτη**.
2. Επίσης, ο τελεστής **&** αν εφαρμοστεί σε μία μεταβλητή δίνει την **διεύθυνση** της μεταβλητής αυτής.

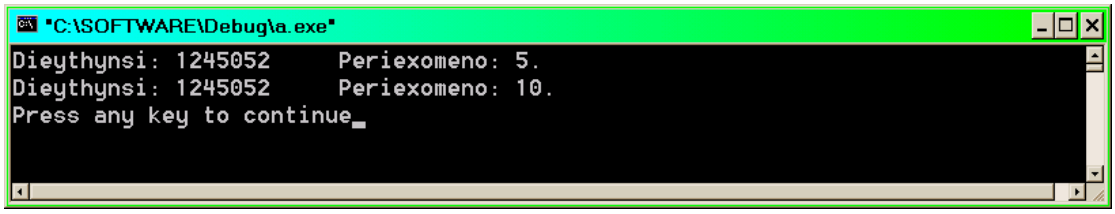
Σημείωση

Κάθε δείκτης περιορίζεται στο να δείχνει σε έναν συγκεκριμένο τύπο δεδομένων. Μοναδική εξαίρεση είναι ο τύπος void, ο οποίος χρησιμοποιείται για να δείχνει οποιονδήποτε τύπο δεδομένων, αλλά και πάλι δεν μπορεί να χρησιμοποιηθεί για έμμεση αναφορά, αλλά πρέπει να μετατραπεί (με typecasting) σε άλλο τύπο δείκτη.*

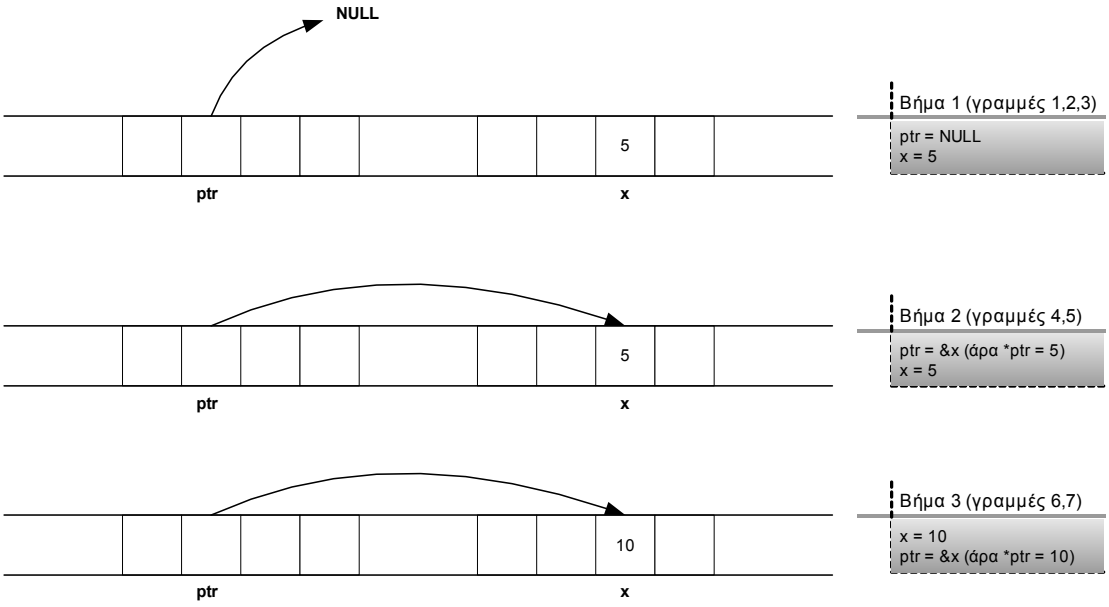
Έστω το ακόλουθο τμήμα κώδικα:

```
1. int x;
2. int *ptr;
3. x = 5;
4. ptr = &x; /* o deiktis ptr deixnei ston x */
5. printf("Dieythynsi: %d\tPeriexomeno: %d.\n",ptr,*ptr);
6. x = 10;
7. printf("Dieythynsi: %d\tPeriexomeno: %d.\n",ptr,*ptr);
```

Στην γραμμή 1 ορίζεται μία μεταβλητή ακεραίου, ενώ στην γραμμή 2 ένας δείκτης σε ακέραιο με όνομα ptr. Στην γραμμή 3 ανατίθεται η τιμή 5 στην μεταβλητή x. Στην συνέχεια (γραμμή 4) ο δείκτης ptr γίνεται ίσος με την διεύθυνση της μεταβλητής x, δηλαδή «ο ptr δείχνει στην μεταβλητή x». Στην γραμμή 5 τυπώνουμε τον δείκτη ptr (δηλαδή την διεύθυνση στην οποία δείχνει, η οποία είναι ίδια με την διεύθυνση του x). Επίσης, τυπώνεται και το περιεχόμενο του δείκτη (δηλαδή το περιεχόμενο της διεύθυνσης ptr). Έπειτα, στην γραμμή 6 η τιμή της μεταβλητής x γίνεται ίση με 10. Τέλος, στην γραμμή 7 ξανατυπώνουμε την διεύθυνση και το περιεχόμενο του ptr. Το αποτέλεσμα της εκτέλεσης του παραπάνω κώδικα φαίνεται στην ακόλουθη εικόνα:



Όπως βλέπουμε και στην παραπάνω εικόνα, το περιεχόμενο του δείκτη άλλαξε όταν αλλάξαμε την τιμή της μεταβλητής x, χωρίς να επέμβουμε στον ίδιο τον δείκτη! Αυτό εξηγείται σχηματικά στο ακόλουθο σχήμα:



Αρχικά, λοιπόν, ο δείκτης ptr δεν δείχνει πουθενά (λέμε ότι η τιμή του είναι **NULL**), ενώ το x έχει τιμή 5. Στην συνέχεια, το ptr δείχνει στο x. Όταν το x αλλάζει τιμή (γίνεται 10), ο ptr εξακολουθεί να δείχνει στο x, άρα το περιεχόμενό του τώρα θα είναι 10.

Σημείωση

Στο παραπάνω σχήμα, τα τετράγωνα δεν συμβολίζουν κελιά (bytes) αλλά ομάδα κελιών. Συγκεκριμένα, για τύπο ακεραίων, κάθε τετράγωνο αντιστοιχεί σε 4 κελιά (4 bytes). Αυτό εξαρτάται από τον τύπο δείκτη που έχουμε (π.χ. για short θα χρειαζόντουσαν 2 bytes).*

Το ακριβώς αντίθετο (αλλά με το ίδιο ακριβώς αποτέλεσμα) συμβαίνει με το ακόλουθο μπλοκ κώδικα:

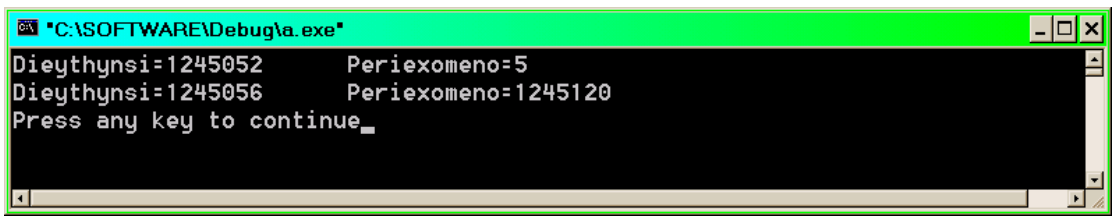
```
int x;
int *ptr;
x = 5;
ptr = &x;
printf("x = %d\n",x);
*ptr = 10;  /* allazoume to periexomeno tou ptr apo 5 se 10 */
printf("x = %d\n",x);
```

Τώρα, δεν αλλάζουμε το x, αλλά το περιεχόμενο του ptr. Όμως, εφόσον το ptr δείχνει στο x, τότε και το x από 5 θα γίνει ίσο με 10.

Ένα άλλο ζήτημα είναι οι πράξεις με δείκτες. Για παράδειγμα, η αύξηση ενός δείκτη κατά 1, κάνει τον δείκτη να δείχνει τόσα κελιά μνήμης πιο μπροστά από το προηγούμενο, όσο είναι το μέγεθος του τύπου. Έστω το ακόλουθο τμήμα κώδικα:

```
int x;
int *ptr;
x = 5;
ptr = &x;
printf("Dieythynsi=%d\tPeriexomeno=%d\n",ptr,*ptr);
ptr++;
printf("Dieythynsi=%d\tPeriexomeno=%d\n",ptr,*ptr);
```

Η εκτέλεση του παραπάνω μπλοκ κώδικα είναι η εξής:



Όπως βλέπουμε, η διεύθυνση ***αυξάνεται κατά 4*** (όχι κατά 1), όσο το μέγεθος του τύπου int. Επίσης, το περιεχόμενο είναι ένας ακαθόριστος αριθμός, δηλαδή ό,τι περιέχει εκείνο το κελί μνήμης (τυχαία) εκείνη την στιγμή.

6.3 ΣΧΕΣΗ ΔΕΙΚΤΩΝ – ΠΙΝΑΚΩΝ

Στην C υπάρχει ισχυρός δεσμός ανάμεσα στους δείκτες και τους πίνακες. Επιπλέον, οι δείκτες μπορούν να χρησιμοποιηθούν σαν πίνακες και μάλιστα με δυναμικό μέγεθος. Ας θυμηθούμε πως ορίζουμε έναν πίνακα, για παράδειγμα ακεραίων, 100 θέσεων:
int x[100];

Έτσι, για παράδειγμα ο συμβολισμός x[3] αντιστοιχεί στο στοιχείο που βρίσκεται στην θέση 3 του πίνακα (η μέτρηση όπως έχουμε πει ξεκινάει από το 0). Τα στοιχεία ενός πίνακα, βρίσκονται στην μνήμη σε διαδοχικές θέσεις. Ορίσουμε έναν δείκτη και τον κάνουμε να δείχνει στο $1^{ο}$ στοιχείο του πίνακα:
*int *ptr;*
ptr = &x[0];

τότε το περιεχόμενο του ptr (*ptr) θα είναι ίσο με το x[0]. Εκτός αυτού όμως (εφόσον τα στοιχεία του πίνακα βρίσκονται διαδοχικά στην μνήμη), και το περιεχόμενο του ptr+1 (δηλαδή το *(ptr+1)), θα είναι ίσο με το x[1]. Γενικά, θα ισχύει:

***(ptr+i) = x[i]**

Το όνομα ενός πίνακα (π.χ. του x) είναι συνώνυμο με ***την διεύθυνση του πρώτου του στοιχείου***. Έτσι, η σχέση ptr = &x[0] είναι συνώνυμη με την έκφραση ptr = x.

Εφόσον οι δείκτες όπως είδαμε συμπεριφέρονται και σαν πίνακες, ένα άλλο στοιχείο που πρέπει να προσέξουμε είναι ότι στην C η έκφραση *(ptr+i) για έναν δείκτη ptr είναι συνώνυμη με την έκφραση ptr[i]. Αυτό μας δίνει μεγάλη ευκολία όταν θέλουμε να χρησιμοποιήσουμε έναν δείκτη σαν πίνακα. (*Προσοχή: το αντίθετο δεν ισχύει: δηλαδή για ένα στατικό πίνακα (όπως τον x) δεν μπορούμε να γράψουμε *(x+i)*).

Παραπάνω είδαμε παραδείγματα στο οποία οι δείκτες αρχικοποιούνται με χρήση άλλων μεταβλητών (π.χ. ptr = &x). Αν θέλουμε να αρχικοποιήσουμε τους δείκτες μόνοι μας **ΔΕΝ** φτάνει να αρχικοποιήσουμε μόνο το περιεχόμενό τους. Αυτό που είναι απαραίτητο είναι να τους δοθεί η διεύθυνση στην οποία θα δείχνουν. Έτσι, η έκφραση:
*int *ptr;*
**ptr = 10;*

είναι ΛΑΘΟΣ και θα οδηγήσει σε απροσδιόριστη συμπεριφορά στο πρόγραμμά μας. Για να δοθεί μία διεύθυνση και ορισμένος χώρος μνήμης προς χρήση σε έναν δείκτη πρέπει να γίνει κλήση της συνάρτησης **malloc()**. Η συνάρτηση αυτή δεσμεύει τον επιθυμητό χώρο μνήμης (συνεχόμενα κελιά) και επιστρέφει την διεύθυνση της αρχής του χώρου αυτού. Όταν καλούμε

λοιπόν αυτήν την συνάρτηση λέμε στον compiler ότι για παράδειγμα θέλουμε 100 bytes μνήμης για χρήση. Η malloc() βρίσκει αν υπάρχουν ελεύθερα 100 συνεχόμενα bytes μνήμης και επιστρέφει την διεύθυνση της αρχής των 100 αυτών bytes. Για όρισμα λαμβάνει το πλήθος των bytes μνήμης που χρειαζόμαστε. Για παράδειγμα η έκφραση:

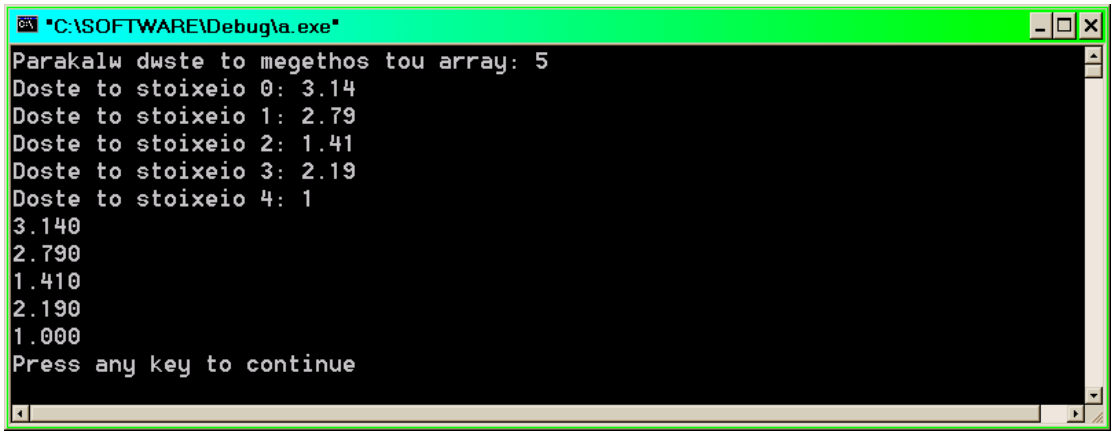
ptr = (int)malloc(sizeof(int)*10);*

δεσμεύει χώρο μνήμης για 10 ακεραίους (δηλαδή 10*4 = 40 bytes) και επιστρέφει την διεύθυνση του 1ου κελιού του χώρου που θα βρει ελεύθερο. Το (int*) μπροστά από την συνάρτηση λειτουργεί ως typecasting για δείκτες και γίνεται γιατί η malloc() κανονικά επιστρέφει δείκτη σε void*, ο οποίος πρέπει να μετατραπεί σε δείκτη σε int με τον τρόπο αυτό. Στο ακόλουθο πρόγραμμα ορίζουμε έναν δείκτη σε float και τον χρησιμοποιούμε σαν array. Στην συνέχεια, διαβάζουμε το μέγεθος του array και έπειτα διαβάζονται όλα τα στοιχεία από το πληκτρολόγιο. Τέλος, εκτυπώνονται τα στοιχεία το ένα κάτω από το άλλο:

```
#include <stdio.h>
int main()
{
	float *array;
	int N,i;
	printf("Parakalw dwste to megethos tou array: ");
	scanf("%d",&N);
	array = (float*)malloc(sizeof(float)*N);
	for (i=0;i<N;i++)
	{
		printf("Doste to stoixeio %d: ",i);
		scanf("%f",&array[i]);
		/*scanf("%f",array+i); */
	}

	for (i=0;i<N;i++)
	{
		printf("%.3f\n",array[i]);
		/*printf("%.3f\n",*(array+i));*/
	}

	return 0;
}
```

Οι 2 γραμμές κώδικα που είναι σε σχόλια, αφορούν τους κλασσικούς τρόπους προσπέλασης δεικτών.

Η εκτέλεση του παραπάνω μπλοκ κώδικα είναι η εξής:


"C:\SOFTWARE\Debug\a.exe"
Parakalw dwste to megethos tou array: 5
Doste to stoixeio 0: 3.14
Doste to stoixeio 1: 2.79
Doste to stoixeio 2: 1.41
Doste to stoixeio 3: 2.19
Doste to stoixeio 4: 1
3.140
2.790
1.410
2.190
1.000
Press any key to continue

7. ΣΥΝΑΡΤΗΣΕΙΣ

Οι συναρτήσεις είναι ίσως το βασικότερο χαρακτηριστικό της γλώσσας C και του δομημένου προγραμματισμού γενικότερα. Κάθε ολοκληρωμένο πρόγραμμα σε C πρέπει να διαχωρίζεται σε συναρτήσεις. Συνήθως, η χρήση των συναρτήσεων αφορά διαδικασίες οι οποίες επαναλαμβάνονται πολλές φορές σε ένα πρόγραμμα.

7.1 ΚΛΗΣΗ ΣΥΝΑΡΤΗΣΕΩΝ – ΠΡΩΤΟΤΥΠΑ ΣΥΝΑΡΤΗΣΕΩΝ

Έχουμε ήδη δει σε προηγούμενα κεφάλαια την κλήση ορισμένων συναρτήσεων. Για παράδειγμα η συνάρτηση abs() της math.h είναι μία συνάρτηση η οποία υπολογίζει την απόλυτη τιμή ενός ακεραίου. Δηλαδή, η συνάρτηση math έχει για ***όρισμα*** έναν ακέραιο αριθμό και ***επιστρέφει*** την απόλυτη τιμή του (δηλαδή έναν άλλο ακέραιο). Από τα παραπάνω καταλήγουμε στον ορισμό του προτύπου μίας συνάρτησης. Το πρότυπο στην ουσία περιγράφει πως καλείται μία συνάρτηση. Αποτελείται από τα εξής 3 στοιχεία:

- Τον τύπο της επιστρεφόμενης τιμής
- Τον αριθμό των παραμέτρων της
- Τον τύπο δεδομένων των παραμέτρων της
- Το όνομά της

Ο τύπος της επιστρεφόμενης τιμής μπορεί να είναι ότι και οι τύποι των μεταβλητών. Επίσης μπορεί να χρησιμοποιηθεί ο τύπος void που σημαίνει ότι η συνάρτηση ***δεν επιστρέφει*** καμία τιμή.

Έτσι, η μορφή των προτύπων είναι η ακόλουθη:

τύπος_επιστροφής όνομα_συνάρτησης(τύπος όνομα_παραμέτρου1
τύπος όνομα_παραμέτρου2
τύπος όνομα_παραμέτρου3
..
..
..
τύπος όνομα_παραμέτρουN);

Για παράδειγμα βλέποντας το πρότυπο:

*int mean_value(int *pinakas, int N)*

καταλαβαίνουμε ότι η συνάρτηση αυτή έχει όνομα mean_value, επιστρέφει έναν ακέραιο και έχει δύο ορίσματα (έναν δείκτη σε ακέραιο και έναν ακέραιο).

Για να καλέσουμε μία συνάρτηση πρέπει να γνωρίζουμε το πρότυπό της και να το ακολουθήσουμε πιστά. Για παράδειγμα, αν θεωρήσουμε δεδομένο το

παραπάνω πρότυπο, τότε μία κλήση της συνάρτησης αυτής θα μπορούσε να ήταν η ακόλουθη:

```
int a, N;
int *p;
…..
…..
a = mean_value(p,N);
```

Παρατηρούμε ότι τα ορίσματα που περνάμε στις συναρτήσεις δεν χρειάζεται να έχουν το ίδιο όνομα με αυτά που έχουν δηλωθεί στο πρότυπο. Εξάλλου στο πρότυπο της συνάρτησης μπορούμε ακόμα και να μην βάλουμε όνομα ορισμάτων, αλλά μόνο τους τύπους τους. Για παράδειγμα:

```
int mean_value(int*, int)
```

7.2 ΟΡΙΣΜΟΣ ΣΩΜΑΤΟΣ ΣΥΝΑΡΤΗΣΕΩΝ

Αφού ορίσουμε το πρότυπο της συνάρτησης, πρέπει να υλοποιήσουμε το ***σώμα*** της, δηλαδή να γράψουμε τον κώδικα που ορίζει βήμα προς βήμα τις ενέργειες της συνάρτησης. Για παράδειγμα, έστω ότι θέλουμε να υλοποιήσουμε μία συνάρτηση που επιστρέφει την μέση τιμή δύο αριθμών:

```
int mean_2(int, int);
int main()
{
    ……
    ……
   return 0;
}
int mean_2(int a, int b)
{
    return (a+b)/2;
}
```

Παρατηρούμε ότι το πρότυπο της συνάρτησης γράφεται πάνω από την main() ενώ ο ορισμός του σώματος της συνάρτησης, μετά την main(). Επίσης, για να επιστρέψουμε την επιθυμητή τιμή χρησιμοποιούμε την εντολή return. Προφανώς αν η συνάρτηση είναι void δεν χρειάζεται να κάνουμε return.

Σημείωση

Στο σώμα μίας συνάρτησης μπορούμε να ορίζουμε μεταβλητές οποιουδήποτε τύπου, όπως ακριβώς κάνουμε και στην main(). ***Η εμβέλεια αυτών των μεταβλητών εκτείνεται μέχρι τα όρια της συνάρτησης****.*

7.3 ΣΥΝΑΡΤΗΣΕΙΣ ΚΑΙ ΔΕΙΚΤΕΣ

Ας υποθέσουμε ότι θέλουμε να γράψουμε μία συνάρτηση η οποία δεν επιστρέφει τίποτα (void), λαμβάνει για όρισμα έναν αριθμό (έστω int) και τον αυξάνει κατά 1. Σαν πρώτη σκέψη θα μπορούσε κανείς απλώς να γράψει:

```
#include <stdio.h>

void add_1(int a);
main()
{
        int x = 2;
        printf("Before add_1: x = %d\n",x);
        add_1(x);
        printf("Add add_1: x = %d\n",x);
}

void add_1(int a)
{
        a++;
}
```

Ωστόσο, αν καλέσουμε αυτή την συνάρτηση θα δούμε ότι μετά την κλήση το περιεχόμενο της μεταβλητής που βάλαμε σαν όρισμα παραμένει το ίδιο!

Αυτό συνέβη διότι στην C τα ορίσματα περνάνε στις συναρτήσεις όχι σαν μεταβλητές αλλά σαν προσωρινά αντίγραφα μεταβλητών. Δηλαδή , στο παραπάνω παράδειγμα όταν καλέσαμε την add_1() με όρισμα την μεταβλητή x, δημιουργήθηκε ένα αντίγραφο της x και η τιμή του όντως αυξήθηκε κατά 1 (αν κάναμε ένα printf στο τέλος του σώματος την συνάρτησης θα το βλέπαμε). Το πρόβλημα είναι ότι μόλις η συνάρτηση επιστρέφει το αντίγραφο αυτό καταστρέφεται.

Για να αντιμετωπιστεί το πρόβλημα αυτό πρέπει να γίνει χρήση δεικτών σαν ορίσματα:

```
#include <stdio.h>

void add_1(int *a);

int main()
{
        int x = 2;
        printf("Before add_1: x = %d\n",x);
```

```
        add_1(&x);
        printf("Add add_1: x = %d\n",x);
        return 0;
}

void add_1(int *a)
{
        (*a)++;
}
```

Στο παραπάνω παράδειγμα, βλέπουμε ότι σαν όρισμα στην συνάρτηση add_1 δεν είναι ένας ακέραιος αλλά ένας δείκτης σε ακέραιο. Έτσι, στο σώμα της συνάρτησης αυξάνουμε κατά ένα το περιεχόμενο της διεύθυνσης στην οποία δείχνει ο δείκτης. Και πάλι, ο ίδιος ο δείκτης δεν μπορεί να αλλάξει μέσα από την συνάρτηση, αλλά αυτό δεν μας ενδιαφέρει στο συγκεκριμένο παράδειγμα. Επίσης, όταν καλούμε την συνάρτηση, δεν δίνουμε σαν όρισμα έναν ακέραιο, αλλά την διεύθυνση του ακεραίου αυτού (χρησιμοποιώντας στον χαρακτήρα &). Αυτό το έχουμε δει και στις κλήσεις της συνάρτησης ***scanf()***.

Συνοψίζοντας λοιπόν, όταν για κάποιο λόγο πρέπει να επέμβουμε στην τιμή κάποιου ορίσματος μίας συνάρτησης χρησιμοποιούμε δείκτες.

7.4 ΧΡΗΣΗ ΟΡΙΣΜΑΤΩΝ ΣΤΗΝ ΣΥΝΑΡΤΗΣΗ MAIN()

Ορισμένα προγράμματα επιτρέπουν το πέρασμα ορισμάτων στην γραμμή εντολής. Τα ορίσματα αυτά ακολουθούν το όνομα του προγράμματος και είναι μεταξύ τους διαχωρισμένα με κενά. Για παράδειγμα ένα πρόγραμμα αντιγραφής αρχείων θα μπορούσε να παίρνει δύο ορίσματα γραμμής εντολής: το όνομα του αρχείου που θα αντιγράψει και το όνομα του αρχείου στο οποίο θα γίνει η αντιγραφή π.χ.:

C:\>copy text.txt newText.txt

Για να καταφέρουμε κάτι τέτοιο με την γλώσσα C χρησιμοποιούμε δύο ειδικές παραμέτρους στην συνάρτηση main():

- **Παράμετρος 1: argc:** αποθηκεύει τον ***αριθμό των ορισμάτων***, είναι τύπου int και έχει πάντα τιμή τουλάχιστον 1 αφού το όνομα του προγράμματος αντιμετωπίζεται σαν όρισμα.
- **Παράμετρος 2: argv:** έχει τύπο **char και αποθηκεύει τα ονόματα των ορισμάτων. Το argv[0] είναι το όνομα του προγράμματος.

Το ακόλουθο παράδειγμα τυπώνει όλα τα ορίσματα που δίνει ο χρήστης:

```
#include <stdio.h>

int main(int argc, char **argv){
        int i;
        for(i=0;i<=argc-1;i++){
                printf("%s\n",argv[i]);
        }
        return 0;
}
```

7.5 ΧΡΗΣΗ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ HEADER ΑΡΧΕΙΩΝ

Έχουμε δει ότι για να μπορέσουμε να κάνουμε κλήση σε ορισμένες συναρτήσεις πρέπει να χρησιμοποιήσουμε το #include. Για παράδειγμα, για να χρησιμοποιήσουμε τις συναρτήσεις printf και scanf πρέπει να κάνουμε include το stdio.h. Αυτό μπορούμε να το κάνουμε γενικά για οποιεσδήποτε συναρτήσεις, κυρίως για ομάδα συναρτήσεων οι οποίες εξυπηρετούν έναν συγκεκριμένο σκοπό. Τα πιο ευρέως χρησιμοποιούμενα header files που υπάρχουν στην C είναι τα ακόλουθα:

stdlib.h: βιβλιοθήκη με τις πιο συνήθεις συναρτήσεις για αλλαγές τύπων, δέσμευση μνήμης και άλλα.

stdio.h: standard βιβλιοθήκη εισόδου / εξόδου

string.h: βιβλιοθήκη για διαχείριση αλφαριθμητικών

time.h: βιβλιοθήκη για συναρτήσεις ώρας / ημερομηνίας

math.h: βιβλιοθήκη για μαθηματικές συναρτήσεις

Τα header αρχεία περιέχουν τα πρότυπα των υλοποιημένων συναρτήσεων. Για κάθε header αρχείο υπάρχει και ένα αρχείο .c το οποίο περιέχει τα σώματα των αντίστοιχων συναρτήσεων. Το αρχείο .c περιέχει στο τμήμα των εντολών προεπεξεργαστή και ένα include στο αντίστοιχο header αρχείο. Η σύνδεση των δύο αρχείων όταν κάνουμε include σε ένα header αρχείο εκτελείται από τον linker.

Έστω για παράδειγμα ότι θέλουμε να δημιουργήσουμε ορισμένες συναρτήσεις για βασική επεξεργασία διανυσμάτων με στοιχεία ακεραίους αριθμούς.

Ακολουθεί ένα παράδειγμα υλοποίησης:

```
/* array1.h */
int *readArray(int N); /* diavasma N stoixeiwn kai epistrofi toy array */
int sumArray(int *array, int N); /* ypologismos ahtroismatos twn stoixevn toy array */
int meanValue(int *array,int N); /* ypologismow mesis timis twn stoixewn toy array */
void printArray(int *array, int N); /* ektypwsi tou array stin othoni */


/* array1.c */
#include <stdio.h>
#include <stdlib.h>
#include "array1.h"

int *readArray(int N)
{
        int i;
        int *array = (int*)malloc(sizeof(int)*N);
        for (i=0;i<N;i++)
                scanf("%d",&array[i]);
        return array;
}

int sumArray(int *array, int N)
{
        int i,sum = 0;
        for (i=0;i<N;i++)
                sum += array[i];
        return sum;
}

int meanValue(int *array,int N)
{
        return sumArray(array,N)/N;
}

void printArray(int *array,int N)
{
        int i;
        for (i=0;i<N;i++)
                printf("%d\n",array[i]);
        return array;
}

/* main.c */
#include <stdio.h>
#include "array1.h"
/* kanoume include sto arxeio twn prototypwn twn synsrtisewn gia ta array */
```

```
int main()
{
	int *A;
	A = readArray(10);
	printf("Sum: = %d\n",sumArray(A,10));
	printf("Mean value = %d\n",meanValue(A,10));
	printArray(A,10);
	return 0;
}
```

Σημείωση 1

Σε ένα header αρχείο εκτός από πρότυπα συναρτήσεων μπορούμε να ορίσουμε και global μεταβλητές, ή και τύπους εγγραφών (records) όπως θα δούμε αργότερα.

Σημείωση 2

Όταν κάνουμε include ένα αρχείο το οποίο δεν ανήκει στην standard βιβλιοθήκη της C (όπως στο παραπάνω παράδειγμα) βάζουμε το όνομα του αρχείου σε διπλά εισαγωγικά, όχι σε < >.

Πώς δημιουργούμε και συμπεριλαμβάνουμε header αρχεία στο πρόγραμμά μας:

1. Στο Visual Studio: Στο τρέχον project κάνουμε File→New στον φάκελο files επιλέγουμε c/c++ header file ή C/C++ source file
2. Στο Unix απλώς δημιουργούμε ξεχωριστά τα αρχεία και στην συνέχεια κάνουμε compile όλα τα .c αρχεία:
   gcc array1.c main.c
   Στην συνέχεια, ο linker αναλαμβάνει να συνδέσει το object αρχείο του array1.c με το κυρίως πρόγραμμα.

8. ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΧΡΗΣΗ ΔΟΜΩΝ

8.1 ΔΗΜΙΟΥΡΓΙΑ ΔΟΜΗΣ (STRUCT)

Όπως έχουμε δει, οι πίνακες επιτρέπουν την ομαδοποίηση ίδιων στοιχείων (π.χ. floats). Ωστόσο, είναι πιθανόν να θέλουμε να ομαδοποιήσουμε δεδομένα τα οποία διαφοροποιούνται μεταξύ τους ως προς τον τύπο. Αυτό επιτυγχάνεται με χρήση του struct:

```
struct struct_name
{
  struct_field1
  struct_field2
   …….
  struct_fieldN
}
```

Έστω για παράδειγμα ότι θέλουμε να φτιάξουμε έναν τύπο ο οποίος περιέχει πληροφορία για τους υπαλλήλους μίας εταιρίας:

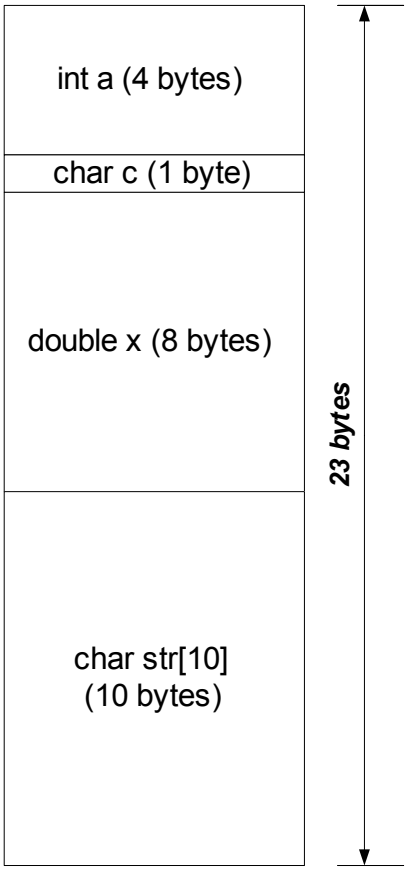
```
struct Employee
{
    char name[50];
    unsigned int age;
    float salary;
    unsigned int code_number;
};
```

Για να δηλώσουμε μία μεταβλητή για τον παραπάνω τύπο γράφουμε για παράδειγμα:
struct Employee employee1;

Για να προσπελάσουμε τα πεδία μίας δομής χρησιμοποιούμε την τελεία (.). Για παράδειγμα, μπορούμε να γράψουμε:

employee1.salary = 1050.5

Το μέγεθος (σε bytes) μίας δομής είναι ίσο με το άθροισμα των μεγεθών των επιμέρους μεταβλητών. Στο ακόλουθο σχήμα φαίνεται ένα παράδειγμα μίας δομής η οποία έχει για πεδία: έναν ακέραιο, έναν χαρακτήρα, έναν double και έναν πίνακα χαρακτήρων 10 θέσεων:



8.2 ΔΕΙΚΤΕΣ ΣΕ STRUCT

Για να ορίσουμε έναν δείκτη σε δομή, κάνουμε ότι και με τα υπόλοιπα είδη μεταβλητών:
*struct structName *varName;*

Για παράδειγμα:
*struct Employee *E*

Για να προσπελάσουμε ένα πεδίο ενός δείκτη σε δομή κάνουμε το εξής:
*(*varName).fieldName;*

Για παράδειγμα:
*(*E).code_number;*

Επειδή οι δείκτες για δομές χρησιμοποιούνται πολύ συχνά, παρέχεται ένας εναλλακτικός τρόπος συμβολισμός σαν συντομογραφία. Αν pVarName είναι δείκτης σε δομή, τότε η έκφραση *pVarName* **->** *fieldname* αναφέρεται στο συγκεκριμένο πεδίο. *Πρέπει να προσέξουμε ότι όταν θέλουμε να περάσουμε μία δομή σαν όρισμα σε μία συνάρτηση και να αλλάξουμε το περιεχόμενο ενός πεδίου, τότε πρέπει να χρησιμοποιήσουμε δείκτη σε δομή (όπως ακριβώς συμβαίνει και με όλα τα είδη μεταβλητών).*

Παραδειγμα της χρησης του τελεστη ->

```
#include <stdio.h>

struct point{
        int x;
        int y;
        struct point *parent;
};

int main()
{
struct point one;
struct point two;

one.x=5;
one.y=7;
one.parent=&two;
(*(one.parent)).y=9;
printf("%i\n",two.y);
one.parent->x=7;
printf("%i\n",two.x);
}
```

Η εκτελεση του παραπανω κωδικα θα μας δωσει :

8.3 ΧΡΗΣΗ ΤΗΣ TYPEDEF

Αν θέλουμε να παραλείψουμε το γράψιμο της λέξης struct στην αρχή κάθε δήλωσης μεταβλητής μίας δομής, πρέπει να χρησιμοποιήσουμε την typedef όταν ορίζουμε την δομή. Με την typedef μπορούμε γενικά να ορίσουμε νέους τύπους δομών κάθε είδους. Για παράδειγμα η δήλωση:
typedef unsigned short Word

κάνει το όνομα Word συνώνυμο του int, οπότε μπορεί να χρησιμοποιηθεί για δήλωση μεταβλητών:
Word W1,W2;

Η χρήση της typedef μπορεί να γίνει και για structs. Για παράδειγμα η δήλωση:

typedef struct EmployeeTag

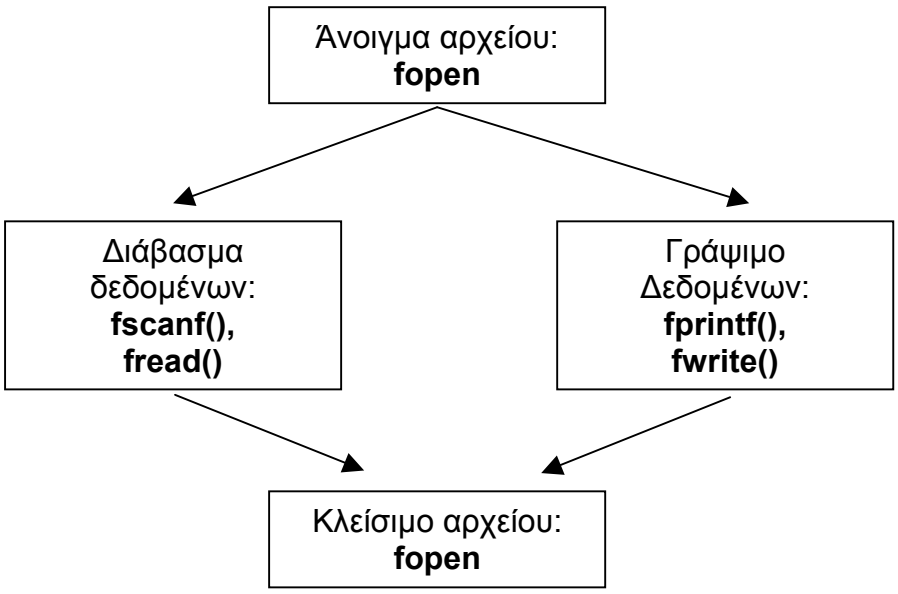
```
{
   char name[50];
   unsigned int age;
   float salary;
   unsigned int code_number;
}Employee;
```

ορίζει έναν νέο τύπο δομής με όνομα Employee. Έτσι, μπορούμε να δηλώνουμε μεταβλητές χρησιμοποιώντας μόνο την λέξη Employee, π.χ. *Employee E*.

9. ΑΡΧΕΙΑ

Τα αρχεία είναι ένα πολύ χρήσιμο μέσο στον προγραμματισμό, εφόσον μας επιτρέπουν να αποθηκεύουμε δεδομένα στον σκληρό δίσκο, χωρίς αυτά να χάνονται μετά την εκτέλεση του προγράμματος. Στο κεφάλαιο αυτό θα δούμε πως μπορούμε να γράψουμε και να διαβάσουμε αρχεία χρησιμοποιώντας εντολές της γλώσσας C.

Στο ακόλουθο σχήμα βλέπουμε μία αναπαράσταση της σειράς των συναρτήσεων που πρέπει να καλέσουμε για να διαβάσουμε ή να γράψουμε δεδομένα από / προς ένα αρχείο:



9.1 ΑΝΟΙΓΜΑ ΚΑΙ ΚΛΕΙΣΙΜΟ ΑΡΧΕΙΩΝ

Για την διαχείριση αρχείων στην C υπάρχει ο τύπος FILE*. Χρησιμοποιώντας αυτόν τον τύπο μπορούμε να γράψουμε και να διαβάσουμε αρχεία.

Για να ***ανοίξουμε*** ένα αρχείο χρησιμοποιούμε την εντολή fopen():

*FILE *fopen(const char *filename, const char *mode);*

Ορίσματα:
filename: το πλήρες όνομα του αρχείου που θέλουμε να ανοίξουμε
mode: ο τρόπος με τον οποίο θέλουμε να ανοιχθεί το αρχείο (π.χ. για διάβασμα ή για γράψιμο)

Επιστρεφόμενη τιμή: Η συνάρτηση επιστρέφει ένα δείκτη σε αρχείο (FILE*)

Στον ακόλουθο πίνακα φαίνονται οι τιμές που μπορεί να λάβει η $2^{η}$ παράμετρος της συνάρτησης:

Mode	Επεξήγηση
“r”	Το αρχείο ανοίγει για διάβασμα. Αν το αρχείο δεν υπάρχει επιστρέφεται η τιμή NULL.
“w”	Το αρχείο ανοίγει για γράψιμο. Αν το αρχείο υπάρχει ήδη, τότε το παλιό αρχείο χάνεται.
“a”	Το αρχείο ανοίγει για γράψιμο στο τέλος του (appending).
“r+”	Το αρχείο ανοίγει για γράψιμο ΚΑΙ για διάβασμα. Αν το αρχείο δεν υπάρχει, επιστρέφεται NULL.
“w+”	Το αρχείο ανοίγει για γράψιμο ΚΑΙ για διάβασμα. Αν το αρχείο υπάρχει, τότε το παλιό αρχείο χάνεται.
“a+”	Ανοίγει το αρχείο για appending και για διάβασμα.

Επίσης, αν στα παραπάνω modes προστεθεί ο χαρακτήρας **b**, τότε το αρχείο ανοίγεται σε **binary mode**. Αν δεν υπάρχει ο χαρακτήρας b τότε το αρχείο ανοίγεται σε ASCII mode.

Για να κλείσουμε ένα αρχείο, πρέπει να χρησιμοποιήσουμε την συνάρτηση fclose() η οποία ως όρισμα λαμβάνει ένα δείκτη σε FILE.

Σημείωση 1

Το FILE ΔΕΝ αντιμετωπίζεται όπως οι υπόλοιποι δείκτες. Στην πραγματικότητα είναι μία εσωτερική δομή της C. Έτσι **δεν** μπορούμε να κάνουμε για παράδειγμα fp++ (όπου fp ένα FILE*), για να προχωρίσουμε τον δείκτη στην επόμενη θέση. Τον σκοπό αυτό εξυπηρετεί η fseek().*

Σημείωση 2

Η fclose() θα οδηγήσει σε σφάλμα αν το όρισμα που θα της δώσουμε είναι NULL. Για τον σκοπό αυτό πρέπει να γίνεται έλεγχος:

```
FILE *fp;
fp = fopen(filename,”r”);
if (fp!=NULL)
{
   ….
   ….
  fclose(fp);
}
else
{
   printf(“Το αρχείο δεν υπάρχει!\n”);
}
```

9.2 ΔΙΑΒΑΣΜΑ ΚΑΙ ΓΡΑΨΙΜΟ ΑΡΧΕΙΩΝ

9.2.1 ΣΥΝΑΡΤΗΣΕΙΣ FPRINTF() ΚΑΙ FSCANF()

Οι συναρτήσεις fprintf και fscanf γράφουν και διαβάζουν αντίστοιχα δεδομένα σε / από αρχεία χρησιμοποιώντας ένα δεδομένο format κειμένου, *όπως ακριβώς και οι συναρτήσεις printf και scanf*. Μοναδική διαφορά είναι ότι οι συναρτήσεις printf και scanf λαμβάνουν ένα επιπλέον όρισμα: *έναν δείκτη αρχείου* (ο οποίος είναι και το 1^{ο} όρισμα των συναρτήσεων).

Παράδειγμα 1 (fprintf):

```
int a=2;
float x=0.2,y=0.1;
FILE *fp;
fp = fopen("test.txt","w");
fprintf(fp,"%d %f %f\n",a,x,y);
fclose(fp);
```

Το παραπάνω πρόγραμμα δημιουργεί ένα αρχείο με όνομα test.txt και γράφει τα περιεχόμενα των μεταβλητών a, x, y διαχωρισμένα μεταξύ τους με κενά. Στο τέλος, γράφει και τον χαρακτήρα αλλαγής γραμμής.

Παράδειγμα 2 (fscanf):

```
int a;
float x,y;
FILE *fp;
fp = fopen("test.txt","r");
if (fp)
{
        fscanf(fp,"%d %f %f\n",&a,&x,&y);
        fclose(fp);
}
else
        printf("To arxeio den yparxei!\n");
```

Προσέχουμε ότι στο 2^{ο} παράδειγμα, γίνεται πρώτα έλεγχος αν το fp είναι NULL ή όχι, δηλαδή αν το αρχείο που προσπαθούμε να ανοίξουμε για διάβασμα υπάρχει ή όχι. Στο 1^{ο} παράδειγμα δεν πραγματοποιούμε έναν τέτοιο έλεγχο. Ωστόσο, για να είμαστε 100% τυπικοί πρέπει να υπάρχει έλεγχος και στην περίπτωση που προσπαθούμε να γράψουμε δεδομένα σε ένα καινούριο αρχείο. Για παράδειγμα μπορεί να προσπαθήσουμε να δημιουργήσουμε αρχείο σε κάποιον κατάλογο στον οποίο δεν έχουμε δικαιώματα ή σε έναν δίσκο ο οποίος είναι γεμάτος. Στις περιπτώσεις αυτές η fopen() θα επιστρέψει NULL.

Η συνάρτηση fprintf() επιστρέφει το πλήθος των bytes που γράφηκαν στο αρχείο, ενώ η scanf() επιστρέφει το πλήθος των μεταβλητών που τους ανατέθηκε τιμή. Και οι 2 συναρτήσεις μετά την εκτέλεσή τους, προχωράνε τον

δείκτη του αρχείου τόσα bytes, όσο το πλήθος των bytes που διαβάστηκαν / γράφηκαν.

9.2.2 ΣΥΝΑΡΤΗΣΕΙΣ FWRITE() ΚΑΙ FREAD()

Οι συναρτήσεις αυτές είναι λίγο πιο δύσκολες στην χρήση τους από ότι οι προηγούμενες, αλλά δίνουν περισσότερες συνατότητες. Οι συναρτήσεις αυτές δεν γράφουν φορμαρισμένα δεδομένα, αλλά έναν ολόκληρο buffer (δηλαδή μία περιοχή μνήμης) από δεδομένα.

Η συνάρτηση fwrite() έχει προτότυπο:

```
unsigned int fwrite(   const void *buffer,
                       unsigned int size,
                       unsigned int count,
                       FILE *stream)
```

Τα ορίσματα που λαμβάνει η συνάρτηση αυτή είναι τα ακόλουθα:
buffer: Δείκτης στα δεδομένα τα οποία θα γραφούν στο αρχείο.
size: Μέγεθος κάθε «κομματιού» δεδομένων σε bytes
count: Μέγιστο πλήθος δεδομένων που θέλουμε να γραφεί.
stream: Δείκτης του αρχείου στο οποίο θα γραφούν τα δεδομένα

Η συνάρτηση επιστρέφει το πλήθος των αντικειμένων που τελικά γράφηκαν στο αρχείο. Μετά την κλήση της συνάρτησης, ο δείκτης του αρχείου προχωράει τόσο, όσο το μέγεθος των δεδομένων που τελικά γράφηκαν.

Η συνάρτηση fread() έχει προτότυπο:

```
unsigned int fwrite(   void *buffer,
                       unsigned int size,
                       unsigned int count,
                       FILE *stream)
```

Τα ορίσματα που λαμβάνει η συνάρτηση αυτή είναι τα ακόλουθα:
buffer: Δείκτης του buffer στον οποίο θα αποθηκευτούν τα δεδομένα.
size: Μέγεθος κάθε «κομματιού» δεδομένων σε bytes
count: Μέγιστο πλήθος δεδομένων που θέλουμε να διαβαστούν.
stream: Δείκτης του αρχείου από το οποίο θα διαβαστούν τα δεδομένα

Η συνάρτηση επιστρέφει το πλήθος των αντικειμένων που τελικά διαβάστηκαν από το αρχείο. Μετά την κλήση της συνάρτησης, ο δείκτης του αρχείου προχωράει τόσο, όσο το μέγεθος των δεδομένων που τελικά διαβάστηκαν.

Στο ακόλουθο παράδειγμα, δημιουργούμε ένα binary αρχείο και γράφουμε με χρήση της fwrite() το περιεχόμενο μίας double μεταβλητής:

```
double number = 2.1;
FILE *fp;
fp = fopen("text.bin","wb");
fwrite((double*)&number,sizeof(double),1,fp);
fclose(fp);
```

Το (double*) που μπαίνει μπροστά από την διεύθυνση του number γίνεται γιατι η συνάρτηση περιμένει για όρισμα (void*). Για να διαβάσουμε τον αριθμό που γράψαμε πρέπει να χρησιμοποιήσουμε τον ακόλουθο κώδικα (χρήση fread()):

```
double number;
FILE *fp;
fp = fopen("text.bin","rb");
if (fp)
{
    fread((double*)&number,sizeof(double),1,fp);
    fclose(fp);
}
```

Ωστόσο, η μεγάλη χρησιμότητα των συναρτήσεων αυτών είναι ότι μπορούμε να γράψουμε και να διαβάσουμε σε / από αρχεία δεδομένα τα οποία είναι ομαδοποιημένα (π.χ. σε πίνακες ή σε δομές). Για παράδειγμα, το ακόλουθο τμήμα κώδικα γράφει το διάνυσμα array στο αρχείο data.bin

```
int N = 10;
double *array = (double*)malloc(sizeof(double)*N);
FILE *fp;
......
......
......
fp = fopen("data.bin","wb");
fwrite((double*)array,sizeof(double),N,fp);
fclose(fp);
```

Για να διαβάσουμε τα δεδομένα του αρχείου που δημιουργήσαμε πρέπει να εκτελέσουμε τον ακόλουθο κώδικα:

```
int N = 10;
double *array = (double*)malloc(sizeof(double)*N);
FILE *fp;
int i;
fp = fopen("data.bin","rb");
if (fp)
{
    fread((double*)array,sizeof(double),N,fp);
    fclose(fp);
}
```

Ακόμα πιο χρήσιμες είναι οι συναρτήσεις αυτές όταν θέλουμε να γράψουμε σε αρχείο δεδομένα συγκεκριμένων τύπων. Έστω για παράδειγμα ότι έχουμε ορίσει τον τύπο:

```
typedef struct stoiximaS
{
        char omada1[15];
        char omada2[15];
        float apodosi1;
        float apodosiX;
        float apodosi2;
        char date_time[15];
} stoixima;
```

και έστω S μία εγγραφή του τύπου αυτού. Τότε, για να γράψουμε την εγγραφή αυτή σε ένα αρχείο κάνουμε:

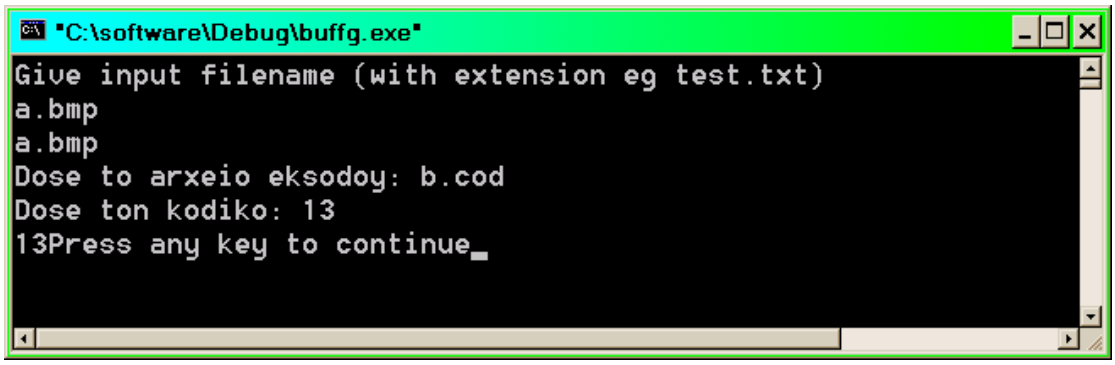
```
fp = fopen("agwnas1.dat","wb");
fwrite((stoixima*)&S, sizeof(stoixima),1,fp);
fclose(fp);
```

Παραδειγμα χρησης των αρχειων οπου ανοιγει αρχειο σε δυαδικη μορφη , δημιουργειτε ένα καινουριο και αντιγραφονται τα bit του ενός στο άλλο με την αλλαγη ενός κωδικου:

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


void main()
{
        FILE *src,*dst;
        int nin,nout,i=0,code;
        unsigned int matr[1024];
   char onoma[20];
printf ("Give input filename (with extension eg test.txt)\n");
scanf("%s",onoma);
printf( "%.20s\n", onoma );
src=fopen(onoma,"rb");
printf ("Dose to arxeio eksodoy");
scanf("%s",onoma);
dst=fopen(onoma,"w+b");
printf ("Dose ton kodiko");
scanf("%d",&code);
printf("%d",code);while(1)
{
nin=fread(matr,sizeof(char),8,src);
for(i=0;i<nin;i++)
matr[i]=matr[i]+code;
```

```
if (feof(src))
fseek(src,sizeof(char),SEEK_CUR);
nout=fwrite(matr,sizeof(char),nin,dst);
if (nin==0)
        break;
}
matr[0]='EOF';
fwrite(matr,sizeof(char),1,dst);
fclose(src);
fclose(dst);
}
```

Το αποτελεσμα της εκτ΄΄ελεσης φαινεται παρακατω:



9.3 ΜΕΤΑΚΙΝΗΣΗ ΔΕΙΚΤΗ ΑΡΧΕΙΟΥ

Όπως είδαμε, χρησιμοποιώντας τις συναρτήσεις fprintf(), fwrite(), fscanf() και fread() ο δείκτης του αρχείου μετακινείται αντίστοιχα. Ωστόσο, υπάρχει περίπτωση να θελήσουμε να μετακινήσουμε τον δείκτη σε κάποια συγκεκριμένη θέση του αρχείου, χωρίς να διαβάσουμε ή να γράψουμε δεδομένα. Η διαδικασία αυτή επιτυγχάνεται με την συνάρτηση fseek(). Το προτότυπο της συνάρτησης αυτής είναι το ακόλουθο:

*int fseek(FILE *stream,*
long offset,
int origin)

Τα ορίσματα που λαμβάνει η συνάρτηση αυτή είναι τα ακόλουθα:
stream: Δείκτης του αρχείου.
offset: Πλήθος bytes μετρούμενα από την θέση origin.
origin: Αρχική θέση.

Η συνάρτηση fseek() μετακινεί τον δείκτη fp offset bytes από την θέση origin. Το origin μπορεί να λαμβάνεις τις εξής σταθερές τιμές (όπως ορίζονται στο stdio.h):

Τιμή origin	**Επεξήγηση**
SEEK_CUR	Τρέχουσα θέση του δείκτη

SEEK_END	Τέλος του αρχείου
SEEK_SET	Αρχή του αρχείου

Για παράδειγμα:

Κώδικας	**Επεξήγηση**
fseek(fp, 10, SEEK_SET)	Ο δείκτης μετακινείται 10 bytes μετά την αρχή του αρχείου.
fseek(fp, 0, SEEK_SET)	Ο δείκτης μετακινείται στην αρχή του αρχείου.
fseek(fp, 2, SEEK_CUR)	Ο δείκτης μετακινείται 2 bytes μετά την τρέχουσα θέση του.
fseek(fp, 10, SEEK_END)	Ο δείκτης μετακινείται 10 bytes πριν το τέλος του αρχείου.
fseek(fp, 0, SEEK_END)	Ο δείκτης μετακινείται στο τέλος του αρχείου

Η συνάρτηση επιστρέφει 0 σε περίπτωση επιτυχίας και μη μηδενική τιμή σε περίπτωση αποτυχίας.