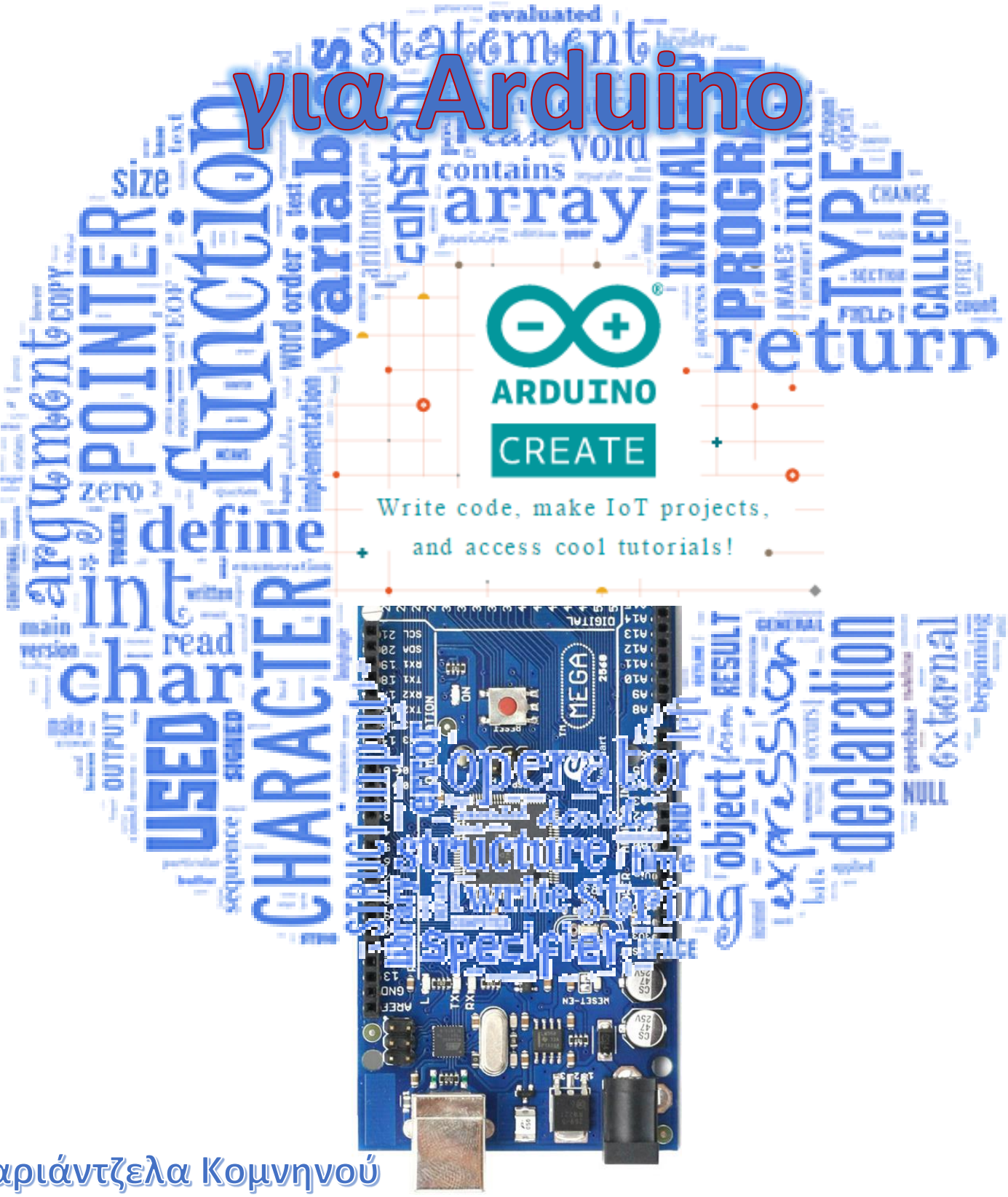
Τα βασικά της C για Arduino



Μαριάντζελα Κομνηνού

ΙΤΥΕ-ΔΙΟΦΑΝΤΟΣ

Η προσπάθεια αυτή έγινε στο πλαίσιο του προγράμματος πρακτικής άσκησης φοιτητών του Πανεπιστημίου Πατρών ως φοιτήτρια του Τμήματος της Επιστήμης των Υλικών. Η πρακτική έγινε στο Ινστιτούτο Τεχνολογίας Υπολογιστών με συνεργασία των Διευθύνσεων Τηλεματικής & Εφαρμογών Περιφερειακής Ανάπτυξης, Εκπαιδευτικής Τεχνολογίας και Τεχνικής Υποστήριξης Συστημάτων.

Η επίβλεψη της πρακτικής άσκησης έγινε από τον Διευθυντή της Διεύθυνσης Τηλεματικής και Εφαρμογών Περιφερειακής Ανάπτυξης καθηγητή του Τμ. Μηχανικών Η/Υ και Πληροφορικής κ. Γιάννη Γαροφαλάκη τον οποίο ευχαριστώ θερμά για την ευκαιρία που μου έδωσε να εκπαιδευτώ σε ένα δομημένο και επαγγελματικό περιβάλλον ανάμεσα σε εξαιρετικούς επιστήμονες.

Επίσης ευχαριστώ τα στελέχη των Διευθύνσεων του ΙΤΥΕ που με βοήθησαν στην μελέτη του υλικού, στην διόρθωση και μορφοποίηση των κειμένων, στην σύνδεση των υλικών, στην παροχή εξοπλισμού, και στην μεταφορά τεχνογνωσίας στην αξιοποίηση λογισμικού σχεδίασης, σε λογισμικό προσομοίωσης, σε πληθώρα βάσεων δεδομένων με αισθητήρες και μικροηλεκτρονικά.

Κατά τη διάρκεια της πρακτικής άσκησης ο σκοπός μας ήταν να δημιουργηθούν οι πρώτες σημειώσεις/οδηγοί για την αξιοποίηση των Edulabs που φτιάχτηκαν πιλοτικά για τα Δημόσια Σχολεία. Το υλικό θα αναρτηθεί στο Αποθετήριο Εφαρμογών του Υπουργείου Παιδείας και Θρησκευμάτων και των Εποπτευόμενων Φορέων (https://github.com/itminedu).

Πρώτα δημιουργήσαμε ένα οδηγό για τα Βασικά της γλώσσας C στον προγραμματισμό της πλατφόρμας μικροϋπολογιστών της οικογένειας των Arduino. Για τον προγραμματισμό και τα παραδείγματα χρησιμοποιήθηκαν ανοιχτά/ελεύθερα λογισμικά όπως ο compiler TDM-GCC MinGW (https://sourceforge.net/projects/tdm-gcc/) και το περιβάλλον Code::Blocks v.16.01 (http://www.codeblocks.org/ Open source, cross platform, free C, C++ and Fortran IDE).

Κατόπιν με ένα δεύτερο οδηγό περιγράψαμε τις ιδιαίτερες πτυχές του προγραμματισμού σε Arduino IDE (www.arduino.cc), τα βασικά ηλεκτρονικά και ηλεκτρολογικά υλικά που χρησιμοποιούνται στην πλατφόρμα και δημιουργήσαμε παραδείγματα χρήσης/υλοποίησης με αυξανόμενο βαθμό δυσκολίας.

Έπειτα, δημιουργήθηκε ένας Οδηγός «Ξεκινήστε γρήγορα με το Arduino» στον οποίο είναι συγκεντρωμένες οι εντολές της C, του Arduino και βασικά ηλεκτρονικά.

Τέλος με τα στοιχεία αυτά φτιάχτηκαν οδηγίες υλοποίησης για 4 παραδείγματα / πειράματα στην πράξη.

Μέσα από την πρακτική άσκηση βελτίωσα τις γνώσεις μου, αξιοποιώντας αυτά που έμαθα στη σχολή μου υπό την οπτική του μηχανικού, του προγραμματιστή και του εκπαιδευτή.

Ευχαριστώ ιδιαίτερα τον καθηγητή μου κ. Δημήτρη Αλεξανδρόπουλο στο Τμήμα Επιστήμης των Υλικών και τον καθηγητή Γιάννη Γαροφαλάκη που με επέβλεπαν ως υπεύθυνοι της πρακτικής εργασία μου στο ΙΤΥΕ.

Μαρία-Αγγελική Κομνηνού

Πάτρα, Δεκέμβριος 2016-Φεβρουάριος 2017

Περιεχόμενα

1. Εισαγωγή στη C

Η γλώσσα C είναι ευρέως χρησιμοποιούμενη σε πολλές εφαρμογές ανάμεσα στις οποίες είναι και η πλατφόρμα με την ευρύτερη ονομασία Arduino. Όταν λέμε Arduino αναφερόμαστε σε μικροεπεξεργαστές διαφορετικών δυνατοτήτων και κατασκευαστών πάνω στους οποίους συνδέονται αξιόπιστοι αλλά φτηνοί αισθητήρες μέτρησης για πολλές διαφορετικές χρήσεις και χρησιμοποιείται η γλώσσα C για να κάνουμε τον μικροεπεξεργαστή να μετρήσει και να αντιδράσει ανάλογα με τις μετρήσεις.

Παρακάτω θα αναφερθούμε στα βασικά στοιχεία και εντολές της C που είναι οι απολύτως απαραίτητες για τον προγραμματισμό σε Arduino.

2. Μεταβλητές

Μεταβλητή ονομάζεται μία θέση μνήμης στον επεξεργαστή που έχει ένα συγκεκριμένο όνομα. Η τιμή μίας μεταβλητής είναι το περιεχόμενο αυτής της θέσης μνήμης (ή των θέσεων μνήμης, όπως θα δούμε) και μπορεί να αλλάξει κατά τη διάρκεια εκτέλεσης του προγράμματος.

Κανόνες για τα Ονόματα Μεταβλητών

Απαράβατοι κανόνες κατά τη δήλωση του ονόματος μίας μεταβλητής:

- Αποτελείται από πεζά και κεφαλαία γράμματα του λατινικού αλφαβήτου και ψηφία.
- Επιτρέπεται ο χαρακτήρας υπογράμμισης '_' (underscore).
- Ο πρώτος χαρακτήρας πρέπει να είναι γράμμα ή ο χαρακτήρας υπογράμμισης '_' .
- Η γλώσσα C είναι case sensitive (δηλ. κάνει διάκριση μεταξύ των πεζών και κεφαλαίων γραμμάτων) συνεπώς, η μεταβλητή με το όνομα nick είναι διαφορετική από τη μεταβλητή με το όνομα Nick.
- Οι δεσμευμένες λέξεις της C απαγορεύεται να χρησιμοποιηθούν ως ονόματα μεταβλητών.

Δήλωση & Εκχώρηση Τιμών σε Μεταβλητές

Για να χρησιμοποιηθεί μια μεταβλητή μέσα στο πρόγραμμα πρέπει πρώτα να δηλωθεί με τον ακόλουθο τρόπο:

τύπος δεδομένων όνομα μεταβλητής;

Το όνομα μεταβλητής είναι το τυχαίο όνομα που επιλέγει ο προγραμματιστής σύμφωνα με τους κανόνες και τις παρατηρήσεις που αναφέρθηκαν προηγουμένως. Ο τύπος δεδομένων είναι ένας από τους αριθμητικούς τύπους δεδομένων που υποστηρίζει η γλώσσα C.

πχ. **int** a; , **float** b; με **int** να αναφέρεται σε ακέραιους και **float** σε πραγματικούς.

Για να εκχωρηθούν τιμές σε μεταβλητές ακολουθείται η εξής διαδικασία:

Παράδειγμα

```
int a;
a = 100;
ή int a = 100;
```

Παράδειγμα

```
int a = 100, b = 200, c = 300
int a = 100, b = a + 100, c = b + 100;
float d = 1.24;
```

Η τιμή μίας μεταβλητής μπορεί να αλλάζει μέσα στο πρόγραμμα. Όταν γίνεται χρήση μίας μεταβλητής στο πρόγραμμα χρησιμοποιείται πάντα η τελευταία τιμή της και όχι κάποια από τις προηγούμενες τιμές της.

Παράδειγμα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int a;
7       a = 1;
8       a = 2;
9       a = 3;
10      printf("Val = %d\n", a);
11      return 0;
12  }
13
```

Έξοδος:

```
Val = 3

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Τύποι Μεταβλητών

Στον παρακάτω πίνακα φαίνονται οι τύποι μεταβλητών στην γλώσσα C. Για κάθε τύπο αναφέρεται το μέγεθος σε bytes Που χρειάζεται για την αποθήκευση της και το εύρος τιμών που μπορεί να έχει μία ανάθεση μεταβλητής. Τέλος για τις περιπτώσεις πραγματικών αριθμών δίνεται και η ακρίβεια κάθε τύπου.

Χρειάζεται προσοχή στις περιπτώσεις που μία ανάθεση είναι στο όριο τιμών που δέχεται ο τύπος. Για παράδειγμα η δήλωση short a = 32767; έχει αναθέση στην μεταβλητή a την μέγιστη τιμή της. Οπότε μετά την εντολή a++; η a θα είναι ίση με -32768.

ΤΥΠΟΣ	ΜΕΓΕΘΟΣ (bytes)	ΕΥΡΟΣ ΤΙΜΩΝ (min-max)	ΨΗΦΙΑ ΑΚΡΙΒΕΙΑΣ
char	1	-128…127	
byte	1	0…255	
int[1]	2	-32.768…32.767	
long	4	-2.147.483.684…2.147.483.647	
float	4	Χαμηλότερη θετική τιμή: $1{,}17*10^{-38}$ Υψηλότερη θετική τιμή: $3{,}4*10^{38}$	6
double	8	Χαμηλότερη θετική τιμή: $1{,}17*10^{-308}$ Υψηλότερη θετική τιμή: $3{,}4*10^{308}$	15
unsigned char	1	0…255	
unsigned int	2	0…65535	
unsigned long	4	0…4.294.967.295	

[1] Στο Arduino το μέγεθος του int είναι 2 bytes, ενώ σε Intel PC και GCC compiler είναι 4 bytes. Παρόμοιες αλλαγές υπάρχουν και για long, κλπ

Δεσμευμένες Λέξεις της C

Δεσμευμένες λέξεις είναι αυτές που τα ονόματά τους χρησιμοποιούνται από τη γλώσσα C και δεν μπορούν να οριστούν μεταβλητές με αυτό το όνομα.

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

Σταθερές

Οι σταθερές είναι μεταβλητές των οποίων οι τιμές δεν αλλάζουν μέσα στο πρόγραμμα και για να δηλωθούν πρέπει να προηγηθεί η λέξη **const** πριν από τον τύπο κάθε μεταβλητής. Επίσης, μαζί με τη δήλωση της σταθεράς, πρέπει να της εκχωρηθεί και μία αρχική τιμή, η οποία δεν θα μπορεί να αλλάξει μέσα στο πρόγραμμα.

Για παράδειγμα εάν a ακέραια σταθερά και θέλουμε να της δώσουμε την τιμή 20 τότε:
const int a = 20;

Εναλλακτικός τρόπος για τη δήλωση μίας σταθεράς είναι η χρήση της οδηγίας #define, η οποία χρησιμοποιείται για τη δήλωση μακροεντολών. Συνήθως, μία μακροεντολή αντιστοιχίζει ένα συμβολικό όνομα με κάποια αριθμητική τιμή. Για τη δήλωση μακροεντολών, η οδηγία #define χρησιμοποιείται ως εξής:

#define όνομα μακροεντολής τιμή

Παράδειγμα: η εντολή: **#define NUM 100** δηλώνει τη μακροεντολή με όνομα NUM και τιμή 100 οπότε όταν ο μεταγλωττιστής όταν συναντάει τη NUM μέσα στο πρόγραμμα την αντικαθιστά με την τιμή 100.

Οι δηλώσεις των μακροεντολών με την οδηγία #define είναι προτιμότερο να γίνονται πριν από τη συνάρτηση main() και τα ονόματά τους με την οδηγία #define είναι προτιμότερο να δηλώνονται με κεφαλαία γράμματα.

⚠ Στο τέλος της οδηγίας **#define** δεν μπαίνει ελληνικό ερωτηματικό (;).

Παράδειγμα

```c
#include <stdio.h>
#include <stdlib.h>
#define NUM 100
int main()
{
    int a, b, c;
    a = 20 - NUM;
    b = 20 + NUM;
    c = 3 * NUM;
    return 0;
}

```

Στο παράδειγμα ι μεταβλητές έχουν τελική τιμή: a = -80 b=120 c=300

3. Συναρτήσεις εξόδου και εισόδου μεταβλητών

Σε ένα πρόγραμμα που εκτελείται χρειάζεται τις περισσότερες φορές να υπάρχει επικοινωνία του προγράμματος με τον χρήστη. Η επικοινωνία αυτή κυρίως γίνεται με την εμφάνιση στην οθόνη πληροφοριών από το πρόγραμμα και την λήψη δεδομένων του χρήστη από το πληκτρολόγιο. Συνήθως οι πληροφορίες που χρειάζεται να εμφανίσουμε είναι τιμές κάποιων μεταβλητών, ενώ στο πρόγραμμα εισάγουμε δεδομένα με την ανάγνωση τιμών μεταβλητών από το πληκτρολόγιο. Αυτά γίνονται όταν φτιάχνουμε προγράμματα για χρήση στον υπολογιστή. Στην περίπτωση του Arduino συνήθως δεν υπάρχει οθόνη ή πληκτρολόγιο. Χρησιμοποιείται αντί αυτών η σειριακή θύρα του τόσο για είσοδο όσο και για έξοδο. Παρόλα αυτά θα παρουσιαστούν οι συναρτήσεις που χρησιμοποιούνται στην γλώσσα C. Εξάλλου η printf έχει παρόμοια εντολή στο Arduino με την ίδια σύνταξη, ενώ η scanf παρουσιάζεται για λόγους πληρότητας στον προγραμματισμό στη γλώσσα C.

1.1 Συνάρτηση printf ()

Η συνάρτηση printf() χρησιμοποιείται για την εμφάνιση δεδομένων σε ένα αρχείο εξόδου stdout (standard output stream) , το οποίο εξ' ορισμού συνδέεται με την οθόνη.

Η συνάρτηση printf() έχει τη μορφή

```
printf("format", var1, var2,…);
```

και δέχεται μία μεταβλητή λίστα παραμέτρων.

- Η πρώτη παράμετρος ("format") είναι ένα αλφαριθμητικό μορφοποίησης (format string), δηλαδή μία ακολουθία χαρακτήρων μέσα σε διπλά εισαγωγικά (" ") η οποία καθορίζει τον τρόπο με τον οποίο θα εμφανιστούν τα δεδομένα στην οθόνη.
- Οι επόμενες παράμετροι είναι προαιρετικές και, αν υπάρχουν, δηλώνουν μεταβλητές που πρέπει να τυπωθούν και η printf() εμφανίζει τις τιμές τους στην οθόνη.

Το αλφαριθμητικό μορφοποίησης (format string) μπορεί να περιέχει:

- Απλούς χαρακτήρες (οι οποίοι εμφανίζονται όπως είναι στην οθόνη)
- Ακολουθίες Διαφυγής
- Προσδιοριστικά Μετατροπής

Παράδειγμα

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

int main()
{
    printf ("This is\n");
    printf ("another C\n");
    printf ("program\n");
    return 0;

}
```

Ενώ εναλλακτικά θα μπορούσαμε να γράψουμε τα κάτωθι, τα οποία θα έβγαζαν το ίδιο αποτέλεσμα:

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

int main()
{
    printf ("This is\nanother C\nprogram\n");
    return 0;
}
```

Και στις δύο περιπτώσεις η **έξοδος** των προγραμμάτων είναι ίδια:

```
This is
another C
program

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Απλοί Χαρακτήρες

Είναι οι χαρακτήρες που θέλουμε να τυπωθούν πριν την εκτύπωση των μεταβλητών ως διευκρίνιση του τι τυπώνεται π.χ. printf("SYNOLO ");

Ακολουθίες Διαφυγής

Μία ακολουθία διαφυγής (escape sequence) χρησιμοποιείται ανάμεσα στα εισαγωγικά " " μαζί με τους απλούς χαρακτήρες είτε για να μετακινηθεί ο δρομέας (cursor) σε κάποια θέση της οθόνης είτε για την εμφάνιση κάποιων ειδικών χαρακτήρων. Αποτελούνται από μια ανάστροφη κεκλιμένη κι έναν ειδικό χαρακτήρα.

Στον παρακάτω πίνακα παρουσιάζονται οι πιο συνηθισμένες εξ' αυτών.

ΑΚΟΛΟΥΘΙΑ ΔΙΑΦΥΓΗΣ	ΣΗΜΑΣΙΑ
\a	Δημιουργία ηχητικού σήματος.
\b	Διαγραφή του τελευταίου χαρακτήρα(= Backspace).
\n	Αλλαγή γραμμής(=Enter).
\r	Επαναφορά του δρομέα(κέρσορα) στην αρχή της τρέχουσας γραμμής.
\t	Μετακίνηση του δρομέα σε απόσταση ίση με το μήκος του Tab.
\\	Εμφάνιση ανάστροφης κεκλιμένης (\).
\"	Εμφάνιση διπλών εισαγωγικών (").

Προσδιοριστικοί Μετατροπείς

Τοποθετούνται και αυτοί ανάμεσα στα εισαγωγικά " ". Αρχίζουν με το χαρακτήρα % και ακολουθούνται από περισσότερους χαρακτήρες με ειδική σημασία. Ο κάτωθι πίνακας παρουσιάζει τους απλούς χαρακτήρες μετατροπής:

ΧΑΡΑΚΤΗΡΑΣ ΜΕΤΑΤΡΟΠΗΣ	ΣΗΜΑΣΙΑ
c	Εμφάνιση χαρακτήρα που αντιστοιχεί σε ακέραια τιμή.
d ή i	Εμφάνιση ακεραίου.
u	Εμφάνιση μη-προσημασμένου ακεραίου.
f	Εμφάνιση πραγματικού αριθμού. Η εξ' ορισμού ακρίβεια είναι 6 δεκαδικά ψηφία.
s	Εμφάνιση χαρακτήρων ενός αλφαριθμητικού.
e ή E	Εμφάνιση πραγματικού αριθμού σε επιστημονική μορφή.
g ή G	Εμφάνιση πραγματικού αριθμού σε κανονική ή επιστημονική μορφή.
p	Εμφάνιση διεύθυνσης μνήμης σε δεκαεξαδική μορφή.
x ή X	Εμφάνιση μη προσημασμένου ακεραίου σε δεκαεξαδική μορφή.
o	Εμφάνιση μη-προσημασμένου ακεραίου σε οκταδική μορφή.
%	Εμφάνιση χαρακτήρα %.

Παράδειγμα

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

int main()
{
    printf ("\a");
    printf ("This\b is a text\n");
    printf ("This\b\b\b is a text\n");
    printf ("This\t is\t a\t text\n");
    printf ("This is a  \"text\"\n");
    printf ("This is a  \\text\\\n");
    printf ("Sample\rtext\n");
    return 0;

}
```

Έξοδος

```
Thi is a text
T is a text
This     is      a       text
This is a  "text"
This is a  \text\
textle

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

3.2 Η Συνάρτηση scanf ()

Η είσοδος δεδομένων στο Arduino δεν γίνεται με τη συνάρτηση scanf. Αντ' αυτής χρησιμοποιείται η σειριακή πόρτα, αφού δεν υπάρχει πληκτρολόγιο. Παρόλα αυτά κάνουμε την παρουσίασή της για λόγους πληρότητας στην γλώσσα C. Αν κάποιος θέλει μπορεί να προχωρήσει στην επόμενη ενότητα χωρίς πρόβλημα.

Μέσα σε ένα πρόγραμμα χρειάζεται ο χρήστης να εισάγει τα δεδομένα του. Αυτό γίνεται συνήθως από το πληκτρολόγιο για προγράμματα που εκτελούνται σε υπολογιστή που έχει οθόνη. Για να μπορέσουμε να αναθέσουμε τιμή σε μια μεταβλητή όταν το πρόγραμμα τρέχει λέμε πως την διαβάζουμε. Το «διάβασμα» μία μεταβλητής μπορεί να γίνει από την οθόνη, από τη σειριακή πόρτα, από αρχείο, ενώ συνηθέστερο σε προγράμματα υπολογιστή είναι από το πληκτρολόγιο, ενώ στο Arduino χρησιμοποιείται η σειριακή θύρα. Η βασική εντολή διαβάσματος είναι η συνάρτηση **scanf("format", διεύθυνση μνήμης της var).**

Τα ονόματα των μεταβλητών που θέλουμε να διαβαστούν εισάγονται στην scanf() μετά τα διπλά εισαγωγικά (" ") με τη χρήση κόμματος (,) και το σύμβολο **&** εμπρός του ονόματός τους. Αν οι μεταβλητές είναι περισσότερες από μία πρέπει και αυτές να διαχωρίζονται μεταξύ τους με κόμμα (,).

Το σύμβολο & μπροστά από το όνομα της μεταβλητής τοποθετείται γιατί στην scanf αναφέρουμε τη διεύθυνση που αποθηκεύεται η μεταβλητή, που θα τοποθετήσει δηλαδή η scanf αυτό που διάβασε από το πληκτρολόγιο

Ο μεταγλωττιστής αντιστοιχίζει ένα-προς-ένα, από αριστερά προς τα δεξιά, τα ονόματα των μεταβλητών με τα προσδιοριστικά μετατροπής ("format").

Αν τα προσδιοριστικά μετατροπής είναι περισσότερα από τις μεταβλητές, τότε για τα πρόσθετα προσδιοριστικά λαμβάνονται τιμές ανάλογες του τύπου της μεταβλητής που έχει δηλωθεί.

Αντίστοιχα, αν τα προσδιοριστικά μετατροπής είναι λιγότερα από τις μεταβλητές, τότε εμφανίζονται ενδείξεις λάθους στη μεταγλώττιση του προγράμματος.

Η συνάρτηση scanf() χρησιμοποιείται για την είσοδο δεδομένων από ένα αρχείο εισόδου, το οποίο ονομάζεται stdin (standard input stream) και εξ' ορισμού συνδέεται με το πληκτρολόγιο.

Η scanf() δέχεται μία μεταβλητή λίστα παραμέτρων, παρόμοια με την printf(), που σημαίνει ότι η πρώτη παράμετρος είναι ένα αλφαριθμητικό μορφοποίησης (format string), το οποίο, συνήθως, περιέχει μόνο απλά προσδιοριστικά μετατροπής (π.χ. %d για μεταβλητές τύπου int, %f για μεταβλητές τύπου float κτλ), ενώ οι επόμενες προαιρετικές παράμετροι είναι οι διευθύνσεις μνήμης των μεταβλητών στις οποίες θα εκχωρηθούν τα δεδομένα που θα εισάγει ο χρήστης από το πληκτρολόγιο.

Κάθε προσδιοριστικό μετατροπής πρέπει να αντιστοιχεί σε μία διεύθυνση μεταβλητής και η αντιστοίχιση γίνεται ένα προς ένα, ενώ για μεταβλητές τύπου double, χρησιμοποιείται το προσδιοριστικό μετατροπής **%lf** και όχι το **%f**, το οποίο χρησιμοποιείται μόνο για μεταβλητές τύπου float.

Παράδειγμα

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

int main()
{
    int a;
    scanf("%d", &a);
    return 0;
}
```

Το **&** που μπήκε πριν το όνομα της μεταβλητής ονομάζεται **τελεστής διεύθυνσης** και χρησιμοποιείται για να αποθηκευτεί ο αριθμός που θα εισάγει ο χρήστης στη διεύθυνση μνήμης της μεταβλητής **a**.

Παράδειγμα

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

int main()
{
    int a;
    float b;
    scanf("%d%f", &a, &b);
    return 0;
}
```

Η πρώτη παράμετρος της **scanf()** είναι το αλφαριθμητικό μορφοποίησης **%d%f**, ενώ οι επόμενες παράμετροι είναι οι διευθύνσεις μνήμης των μεταβλητών a και b αντίστοιχα ενώ το %d αντιστοιχεί στη διεύθυνση της μεταβλητής a και το %f αντιστοιχεί στη διεύθυνση της μεταβλητής b. Παρατηρείται λοιπόν, ότι η αντιστοίχιση γίνεται ένα προς ένα και από αριστερά προς τα δεξιά.

Η εισαγωγή των διαφορετικών τιμών γίνεται είτε με κενό ανάμεσα στις τιμές ή με αλλαγή γραμμής.

Παράδειγμα

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

int main()
{
    char ch;
     scanf("%c", &ch);
     return 0;
}
```

Σε αυτό το παράδειγμα η scanf() διάβασε έναν χαρακτήρα και τον αποθήκευσε στη μεταβλητή ch.

Παράδειγμα

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

int main()
{
    char str[100];
     scanf("%s", str);
     return 0;
}
```

Σε αυτό το παράδειγμα η scanf() διάβασε ένα αλφαριθμητικό και το αποθήκευσε στον πίνακα χαρακτήρων str. Επίσης, παρατηρείται, ότι πριν από τη μεταβλητή str δεν χρησιμοποιείται ο τελεστής &, γιατί το όνομα ενός πίνακα ισοδυναμεί με τη διεύθυνση του πρώτου στοιχείου του.

Η scanf() απαιτεί τον τελεστή διεύθυνσης **&** πριν από το όνομα κάθε αριθμητικής μεταβλητής (π.χ. int, double, char, float, …) αλλιώς το πρόγραμμά δεν θα εκτελεστεί σωστά, ενώ αντιθέτως, όταν ο τύπος της μεταβλητής είναι δείκτης, ο τελεστής διεύθυνσης δεν χρειάζεται.

Παράδειγμα

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

int main()
{
    int a, b;
     scanf("%d, %d", &a, &b);
     printf("%d %d\n", a, b);
     return 0;
}
```

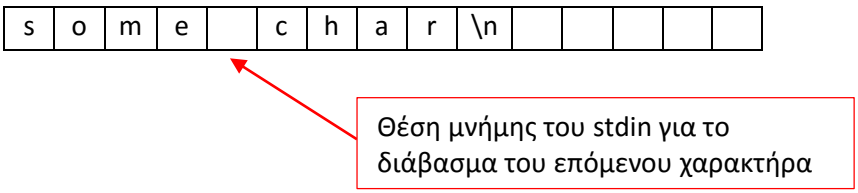
Στο παράδειγμα αυτό διαβάζονται και τυπώνονται δύο ακέραιες μεταβλητές

3.3 Καθαρισμός Μνήμης

Είναι μία καλή τακτική όταν θέλουμε σε ένα πρόγραμμα να ξεκινήσουμε ένα νέο κύκλο στο διάβασμα μεταβλητών να είμαστε σίγουροι πως δεν υπάρχει κάποιος χαρακτήρας που να έχει παραμείνει στο χώρο διαβάσματος (Input buffer). Ένας τρόπος για να αδειάσουμε τη μνήμη του πληκτρολογίου από τα δεδομένα που έχουν παραμείνει είναι με τη χρήση της συνάρτησης **getchar()** χρησιμοποιώντας τον παρακάτω επαναληπτικό βρόχο:

```
while( getchar() != '\n' );
```

Η getchar διαβάζει τον επόμενο χαρακτήρα μέχρι να βρεθεί αλλαγή γραμμής (enter). Με τον τρόπο αυτό στο χώρο (buffer) του standard input (stdin) προχωρά ο δείκτης που δείχνει το επόμενο για διάβασμα διαθέσιμο χαρακτήρα μέχρι το χαρακτήρα \n και στην ουσία το άδειασμα του



3.4 Μετατροπή Τύπου (Type Cast)

Για περιπτώσεις όπου ένας τύπος δεδομένων πρέπει να μετατραπεί προσωρινά σε έναν άλλον τύπο δεδομένων π.χ. μια μεταβλητή που αρχικά είχε δηλωθεί ως int και πρέπει προσωρινά να μετατραπεί σε float ή και το αντίστροφο γράφεται ως εξής:

(τύπος_δεδομένων) (παράσταση)

Παράδειγμα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int a = 10, b = 50;
7       float c;
8       c = (float)a/b;
9       printf ("%.2f\n", c);
10      printf ("%d\n", a);
11      return 0;
12  }
13
```

.

Και η **έξοδος** είναι:

```
0.20
10

Process returned 0 (0x0)   execution time : -0.000 s
Press any key to continue.
```

⚠ Αν στην εντολή c= (float) a/b; Δεν υπήρχε το (float) τότε η διαίρεση a/b θα έδινε σαν διαίρεση δύο ακεραίων το ακέραιο μέρος της διαίρεσης δηλαδή την τιμή 0. Άρα το c = 0

4. Τελεστές

Ο Τελεστής είναι ο τρόπος που μία πράξη εφαρμόζεται σε τιμές μιας ή περισσοτέρων μεταβλητών. Ο Τελεστής συνήθως αναπαρίσταται με ένα σύμβολο το οποίο κάνει (εκτελεί) την πράξη

Τελεστής Εκχώρησης Τιμής (=)

Χρησιμοποιείται για την ανάθεση τιμής σε μια μεταβλητή.

Για παράδειγμα. j = 20; , αυτό σημαίνει ότι η τιμή της μεταβλητής j έγινε 20 ενώ εάν j = z; , τότε η τιμή της j γίνεται ίση με την τιμή της μεταβλητής z.

Παράδειγμα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int a;
7       float b;
8       b = a = 10.22;
9       return 0;
10  }
11
```

Στο παραπάνω παράδειγμα έχουμε μια διπλή ανάθεση **b = a = 10.22;** Η ανάθεση γίνεται από τα δεξιά προς τα αριστερά. Άρα το **a** που είναι ακέραιος λαμβάνει το ακέραιο μέρος του 10.22, άρα την τιμή 10 (**a = 10**) και κατόπιν το **b = a** δηλαδή ο πραγματικός **b** λαμβάνει την τιμή 10.0 (**b = 10.0**)

⚠ Οι εκχωρήσεις σε μία γραμμή γίνονται από δεξιά προς τα αριστερά (π.χ. b = a = 10.22;)

Αν η εκχώρηση ήταν **a = b = 10.22;** τότε το **b = 10.22** και το **a** θα έπαιρνε το ακέραιο μέρος του **b (b = 10.22)** δηλαδή **a = 10**

Αριθμητικοί Τελεστές

Μαθηματικοί Τελεστές των τεσσάρων πράξεων +, -, *, / : Χρησιμοποιούνται για την εκτέλεση των μαθηματικών πράξεων.

Τελεστής **%:** Χρησιμοποιείται για τον υπολογισμό του υπολοίπου της διαίρεσης δυο ακέραιων αριθμών.

⚠ Ο τελεστής % μπορεί να εφαρμοστεί ΜΟΝΟ μεταξύ ακεραίων αριθμών.

Παράδειγμα

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

int main()
{
    int a, b, c;
    a = 13;
    b = 3;
    c = a%b;
    return 0;
}
```

Στο παράδειγμα το **c** λαμβάνει την τιμή **1** που είναι το υπόλοιπο της διαίρεσης του 13/3 (πηλίκο 4 και υπόλοιπο 1).

Ο Τελεστής Αύξησης (++)

Αυξάνει την τιμή μιας μεταβλητής κατά ένα και μπαίνει είτε πριν είτε μετά απ' αυτή.

Παράδειγμα

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

int main()
{
    int a = 2;
    a++;
    printf("Num = %d\n", a);
    return 0;
}
```

Έξοδος:

```
Num = 3

Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```

Ο Τελεστής Αύξησης (++) στην Εκχώρηση

Εάν ο τελεστής ++ χρησιμοποιείται μετά το όνομα της μεταβλητής, τότε πρώτα χρησιμοποιείται η τρέχουσα τιμή της μεταβλητής και μετά αυξάνεται κατά ένα. Αντιθέτως, αν χρησιμοποιείται πριν το όνομα της μεταβλητής, πρώτα αυξάνεται η τιμή της μεταβλητής κατά ένα και μετά χρησιμοποιείται.

Παράδειγμα

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

int main()
{
    int a, b;
    a = 4;
    b = a++;
    printf("a = %d b = %d\n", a, b);
    return 0;
}
```

Έξοδος: a = 5 b = 4

Παράδειγμα

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

int main()
{
    int a, b;
    a = 4;
    b = ++a;
    printf("a = %d b = %d\n", a, b);
    return 0;
}
```

Έξοδος: a = 5 b = 5

Τελεστής Μείωσης (--)

Μπορεί να μπει πριν ή μετά το όνομα μιας μεταβλητής, η οποία σε κάθε περίπτωση μειώνει κατά ένα, ενώ ισχύουν οι ίδιοι κανόνες που εφαρμόζονται και για τον τελεστή αύξησης.

Παράδειγμα

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

int main()
{
    int a = 3;
    a--;;
    printf("Num = %d\n", a);
    return 0;
}
```

Έξοδος:

```
Num = 2

Process returned 0 (0x0)   execution time : 0.021 s
Press any key to continue.
```

Σύνθετοι Τελεστές Ανάθεσης Τιμής (+=, -=, *=, /=)

Πολλές φορές στη C για εξοικονόμηση κώδικα αυξάνουμε την τιμή μιας μεταβλητής με έναν από τους βασικούς αριθμητικούς τελεστές. Έτσι:

a += 10; είναι ισοδύναμο με a = a + 10;
a -= 5; είναι ισοδύναμο με a = a - 5;
a *= (b+2); είναι ισοδύναμο με a = a *(b+2);
a /= b; είναι ισοδύναμο με a = a / b;

Ο Τελεστής sizeof

Ο τελεστής αυτός επιστρέφει το μέγεθος σε bytes του τελεστέου του. Ο τελεστέος μπορεί να είναι το όνομα μιας μεταβλητής ή μπορεί να είναι ένας τύπος. Αν είναι τύπος ονόματος, τότε πρέπει ο τελεστέος να μπει μέσα σε παρενθέσεις, αλλιώς οι παρενθέσεις είναι προαιρετικές.

Παράδειγμα

```c
#include <stdio.h>
main()
{
        int n=10;
        printf("n is %d bytes, every integer is %d bytes. \n", sizeof n, sizeof(int));
        return 0;
}
```

Έξοδος:

```
n is 4 bytes, every integer is 4 bytes.

Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Τελεστές Σύγκρισης (>, >=, <=, <, !=, ==)

Χρησιμοποιούνται για να συγκρίνονται οι τιμές που έχουν δύο εκφράσεις και συνήθως εφαρμόζονται σε εντολές ελέγχου (π.χ. **if**) και σε επαναληπτικούς βρόχους (π.χ. **for**).

ΤΕΛΕΣΤΗΣ	ΣΗΜΑΣΙΑ	ΠΑΡΑΔΕΙΓΜΑ
>	μεγαλύτερο από	if (a > 5)
>=	μεγαλύτερο ή ίσο με	if(a >= 5)
<	μικρότερο από	if(a < 5)
<=	μικρότερο ή ίσο με	if(a <= 5)
!=	διάφορο του	if(a != 5)
==	ίσο με	if(a == 5)

Τα αποτελέσματα έκφρασης στην οποία χρησιμοποιείται τελεστής σύγκρισης είναι **1** όταν η έκφραση είναι **αληθής-true**, ενώ το αποτέλεσμα είναι **0** όταν η έκφραση είναι **ψευδής-false**.

Παράδειγμα:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int a = 3, b = 5;
7       a = (a > 3) + (b <= 5);
8       a = (a == 3) + ((b - 2) >= 3);
9       b = (b != 5);
10      printf("%d %d\n", a,b);
11      return 0;
12  }
13
```

Αρχικά το a = 3 και b = 5

Στην δεύτερη ανάθεση τιμής στο a μετά από τις συγκρίσεις το a >3 είναι false άρα 0 και το b<=5 true γιατί η τιμή είναι 5 άρα 1, άρα το a = 0 + 1 = 1

Στην τρίτη ανάθεση τιμής στο a μετά τις συγκρίσεις προκύπτει a==3 είναι false άρα έχει την τιμή 0 γιατί το a =1 από την προηγούμενη εντολή ανάθεσης και το (b – 2) = (5 – 2) = 3 >= 3 είναι true άρα παίρνει την τιμή 1. Έτσι a = 0 + 1 = 1

Στην Τρίτη ανάθεση το b != 5 είναι false και παίρνει την τιμή 0 επειδή το b = 5 (άρα b !=5 false) και έτσι b = 0.

Έξοδος: 1 0

⚠ Ο τελεστής ελέγχου ισότητας (==) ΔΙΑΦΕΡΕΙ από τον τελεστή εκχώρησης (=)

Ο Τελεστής (!)

Ο τελεστής **!** είναι μοναδιαίος, δηλαδή εφαρμόζεται σε έναν μόνο τελεστέο.

Αν μία έκφραση **exp** είναι αληθής (δηλαδή έχει μη μηδενική τιμή), τότε το αποτέλεσμα της πράξης **!exp** είναι μηδέν (0) ενώ αν είναι ψευδής (δηλαδή έχει μηδενική τιμή), τότε το αποτέλεσμα της πράξης **!exp** είναι ένα (1).

Παράδειγμα

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

int main()
{
    int a = 5;
    printf("Num = %d\n", !a);
    return 0;
}
```

Έξοδος:

```
Num = 0

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Το σύνηθες είναι να χρησιμοποιείται ο τελεστής **!** σε συνθήκες ελέγχου στην εντολή **if**.

π.χ. η if(!a) είναι ισοδύναμη με την if(a == 0)

ενώ η if(a) είναι ισοδύναμη με την if(a != 0)

Λογικοί Τελεστές

- Ο τελεστής **&&**:
  - Η τιμή έκφρασης που περιέχει τον τελεστή **&&** είναι αληθής (παίρνει τιμή 1), ΜΟΝΟ αν όλοι οι όροι είναι αληθείς.
  - Η τιμή έκφρασης που περιέχει τον τελεστή **&&** είναι ψευδής (παίρνει τιμή 0), έστω κι αν ένας όρος είναι ψευδής.
  - Ο τελεστής **&&** εφαρμόζει τη λογική πράξη ΚΑΙ (λογική πράξη AND) μεταξύ των όρων στους οποίους εφαρμόζεται.

- Ο τελεστής ||:
  - Μία έκφραση που περιέχει τον τελεστή || είναι αληθής (παίρνει τιμή 1), αν έστω και ένας όρος της έκφρασης είναι αληθής.
  - Μία έκφραση που περιέχει τον τελεστή || είναι ψευδής (παίρνει τιμή 0), αν κανένας όρος της έκφρασης δεν είναι αληθής.
  - Ο τελεστής || εφαρμόζει δηλαδή τη λογική πράξη Ή (λογική πράξη OR) μεταξύ των όρων στους οποίους εφαρμόζεται

Προτεραιότητα Τελεστών

Η σειρά με την οποία ενεργούν οι τελεστές χωρίς την παρουσία παρενθέσεων είναι:

1. τελεστές αύξησης ή μείωσης ως πρόθεμα (a++, ++a, a--, --a)
2. τελεστές προσαρμογής τύπων ((int), (float))
3. αριθμητικοί τελεστές πολλαπλασιασμού διαίρεσης και ακεραίου υπολοίπου (*,/,%)
4. τελεστές της πρόσθεσης και της αφαίρεσης (+,-)
5. συγκριτικοί τελεστές (==, !=, >, >=, <, <=)
6. λογικοί τελεστές (!, &&, II)
7. τελεστές απόδοσης τιμής (=, +=, -=, *=, /=)

Επειδή είναι πολύ πιθανό να γίνει λάθος στις προτεραιότητες είναι καλό προγραμματιστικά να χρησιμοποιούμε παρενθέσεις που έχουν την πρώτη προτεραιότητα και με αυτό τον τρόπο μπορούμε να ομαδοποιήσουμε τις πράξεις μας.

5. Εντολές Ελέγχου – Απόφασης

Τα προγράμματα συνήθως αλληλοεπιδρούν με το περιβάλλον και τον χρήστη και για διαφορετικά δεδομένα εισόδου εμφανίζουν άλλα δεδομένα εξόδου. Χρειάζεται λοιπόν τρόπος να λαμβάνονται αποφάσεις ανάλογα με τις συνθήκες που ισχύουν και να επιλέγουν την κατάλληλη εκτέλεση κώδικα που πρέπει να εκτελεστεί κάτω από αυτές τις συνθήκες.

Στο πρόγραμμα μία απόφαση που λαμβάνεται μετά από έλεγχο συνθηκών οδηγεί στην αποφυγή εκτέλεσης μέρους του προγράμματος και στην εκτέλεση ενός άλλου τμήματος κώδικα. Η εντολή if είναι η απλούστερη μορφή απόφασης, ενώ η switch δημιουργεί κλάδους για πολλές ομάδες εντολών, ανάλογα με την τιμή μιας μεταβλητής.

5.1 Η Εντολή **if**

Η εντολή **if** είναι μία από τις βασικότερες δομές ελέγχου ροής στη C. Με αυτή γίνεται δυνατή η επιλεκτική εκτέλεση ενός τμήματος κώδικα, ανάλογα με την τιμή μίας συνθήκης.

Παράδειγμα

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

int main()
{
    int x = 3;
    if(x != 0)
    {
        printf("x isn't zero\n");
    }
    return 0;
}
```

Έξοδος:

```
x isn't zero

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Εάν περιέχεται μόνο μια εντολή στο μπλοκ εντολών, τότε τα άγκιστρα μπορούν να παραλειφθούν.

Παράδειγμα

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

int main()
{
    int x = 3;
    if(x > 0)
        printf("x is positive\n");
    return 0;
}
```

Έξοδος:

```
x is positive

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Εάν όμως περιέχονται παραπάνω από μια εντολές, τότε τα άγκιστρα είναι απαραίτητα.

Παράδειγμα

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

int main()
{
    int x = 3;
    if(x > 0)
    {
        printf("x is positive\n");
        printf("In other words, x is greater than zero\n");
    }
    return 0;
}
```

Έξοδος:

```
x is positive
In other words, x is greater than zero

Process returned 0 (0x0)   execution time : -0.000 s
Press any key to continue.
```

⚠ Η εντολή **if** δε δέχεται ερωτηματικό στο τέλος της.

Όταν θέλουμε να προσδιορίσουμε μία ομάδα εντολών που θα εκτελείται όταν μία συνθήκη είναι αληθής (true) και μία άλλη που θα εκτελείται όταν η συνθήκη αυτή είναι ψευδής (false), τότε χρησιμοποιούμε την εντολή ελέγχου **if...else**.

Παράδειγμα

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

int main()
{
    int x = -3;
    if(x > 0)
    {
        printf("x is positive\n");
    }
    else
    {
        printf("x is negative or zero\n");
    }
     return 0;
}

```

Έξοδος:

```
x is negative or zero

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Υπάρχουν περιπτώσεις που οι έλεγχοι μπορούν να γίνουν πιο περίπλοκοι και να έχουμε βρόγχους ελέγχου μέσα σε άλλους βρόγχους ελέγχων. Σε αυτή την περίπτωση μας βοηθούν τα άγκιστρα { για την δήλωση της αρχής και } για το κλείσιμο του βρόγχου.

Επίσης το περιβάλλον που χρησιμοποιούμε για να γράφουμε τον κώδικα μπορεί να μας βοηθήσει βάζοντας σημάδια όπου ανοίγει και κλείνει κάποιος βρόγχος.

Παράδειγμα

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

int main()
{
    int a = 10, b = 20, c = 30;
    if(a > 5)
    {
        if(b == 20)
            printf("1\n");
        if(c == 40)
            printf("2\n");
        else
            printf("3\n");
    }
    else
        printf("4\n");
        return 0;
}
```

Έξοδος:

1 3

Στο παρακάτω παράδειγμα διαβάζεται μια μεταβλητή a.

Ανάλογα με το αν είναι 1 ή 2 τυπώνει την αντίστοιχη λέξη One ή Two διαφορετικά τυπώνει «Something else»

Παράδειγμα

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

int main()
{
    int a;
    printf("Enter number: ");
    scanf("%d", &a);
    if(a == 1)
        printf("One\n");
    else if(a ==2)
        printf("Two\n");
    else
        printf("Something else\n");
    printf("End\n");
    return 0;
}
```

Τελεστής (?:)

Επιτρέπει την εκτέλεση μίας από δύο ενέργειες, σύμφωνα με την τιμή μίας έκφρασης και η σύνταξή του είναι:

```
exp1 ? exp2 : exp3;
```

Αν η έκφραση exp1 είναι αληθής, τότε θα εκτελεστεί η έκφραση που ακολουθεί το ερωτηματικό ? (δηλαδή η exp2), αλλιώς θα εκτελεστεί η έκφραση που ακολουθεί την άνω-κάτω τελεία : (δηλαδή η exp3).

Παράδειγμα

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

int main()
{
    int b = 20;
    (b > 10) ? printf("One\n") : printf("Two");
    return 0;
}
```

Έξοδος:

```
One

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

5.2 Η Εντολή **switch**()

Αν έχουμε ένα πλήθος αποφάσεων που όλες βασίζονται στις τιμές μία μεταβλητής τότε χρησιμοποιούμε την εντολή switch() με την παρακάτω γενική μορφή σύνταξης.

```
switch(n)
  {
    case 1:
          Εντολή;
          Εντολή;
          break;
    case 2:
          Εντολή;
          Εντολή;
          break;
    case 3:
          Εντολή;
          break;
    default:
          Εντολή;
          Εντολή;
}
```

Η switch ακολουθείται από μία μεταβλητή (στο παράδειγμα n) μέσα σε παρενθέσεις. Η μεταβλητή παίζει το ρόλο του διακόπτη (switch) και η απόφαση επιλογής δρόμου βασίζεται αποκλειστικά στην τιμή της.

Η εντολή περικλείεται σε άγκιστρα που οριοθετούν ένα τμήμα προγράμματος με διάφορες περιπτώσεις (case). Κάθε case ακολουθείται από μία σταθερά που δεν είναι σε παρενθέσεις, αλλά συνοδεύεται από άνω και κάτω τελεία. Ο τύπος δεδομένων των σταθερών για κάθε περίπτωση θα πρέπει να συμφωνεί με το τύπο δεδομένων της μεταβλητής του switch.

Όταν η τιμή της μεταβλητής είναι ίδια με την τιμή της σταθεράς μιας εκ των περιπτώσεων τότε εκτελούνται οι εντολές που ακολουθούν την άνω και κάτω τελεία.

Η εντολή **break** εξαναγκάζει τον έλεγχο του προγράμματος να βγει έξω από την πρόταση switch και να συνεχίσει μετά το άγκιστρο που οριοθετεί το τέλος της. Είναι πολύ σημαντικό να μην παραληφθεί, γιατί διαφορετικά θα εκτελεστούν όλες οι εντολές που ακολουθούν μέχρι το άγκιστρο.

Η λέξη **default** παίζει τον ρόλο του else της εντολής if. Με παρόμοιο τρόπο, όταν χρειάζεται να εκτελεστούν κάποιες εντολές, όταν η μεταβλητή διακόπτης δεν ισούται με κάποια σταθερά των περιπτώσεων case, τότε τις γράφουμε μετά τη λέξη default. Είναι με αυτό τον τρόπο οι προκαθορισμένες εντολές που θα εκτελεστούν όταν η τιμή της μεταβλητής δεν αντιστοιχεί σε καμιά από τις σταθερές case.

Παράδειγμα

```
#include<stdio.h>
int main()
{
    int day;
    printf("Give me a number from 1 to 7: ");
    scanf("%d",&day);
    switch(day){
        case 1:  printf("The first day of a week is Monday.\n");    break;
        case 2:  printf("The second day of a week is Tuesday.\n");  break;
        case 3:  printf("The third day of a week is Wednesday.\n"); break;
        case 4:  printf("The fourth day of a week is Thursday.\n"); break;
        case 5:  printf("The fifth day of a week is Friday.\n");    break;
        case 6:  printf("The sixth day of a week is Saturday.\n");  break;
        case 7:  printf("The seventh day of a week is Sunday.\n");  break;
        default: printf("You gave a wrong number. Only from 1 to 7\n");
    }
    return 0;
}
```

Στο παράδειγμα ζητείται η είσοδος ενός αριθμού από το 1 ως το 7και ανάλογα με την είσοδο τυπώνεται και η αντίστοιχη μέρα, ενώ αν δοθεί αριθμός διαφορετικός το πρόγραμμα μας ενημερώνει πως δώσαμε αριθμό.

6. Οι Βρόγχοι Επανάληψης

Οι βρόχοι επανάληψης εφαρμόζονται όταν χρειάζεται ένα μέρος κώδικα ενός προγράμματος να επαναληφθεί μερικές φορές, μέχρι να ισχύσει κάποια συνθήκη ελέγχου. Οι εντολές βρόγχων επανάληψης είναι οι for, while, do-while, ενώ χρησιμοποιούνται και οι εντολές break, continue που θα δούμε παρακάτω.

6.1 Ο Βρόγχος **for**

Η εντολή **for** χρησιμοποιείται για τη δημιουργία επαναληπτικών βρόχων στη C.

Επαναληπτικός βρόχος καλείται το τμήμα του κώδικα μέσα σε ένα πρόγραμμα, το οποίο εκτελείται από την αρχή και επαναλαμβάνεται όσο μία συνθήκη παραμένει αληθής (true).

Γενική Σύνταξη:

```
for( αρχική_έκφραση;  συνθήκη;  τελική_έκφραση )
{
      εντολές
}
```

Βήματα Εκτέλεσης της for

- Εκτελείται η αρχική_έκφραση.
  - Η αρχική_έκφραση εκτελείται μόνο μία φορά, όταν αρχίζει η εκτέλεση της **for** και μπορεί να είναι οποιαδήποτε έγκυρη έκφραση της C.
  - Συνήθως, είναι μία εντολή εκχώρησης που αρχικοποιεί κάποια μεταβλητή, η οποία θα χρησιμοποιηθεί από τις άλλες δύο εκφράσεις.
- Γίνεται έλεγχος της τιμής της συνθήκης.
  - Η συνθήκη είναι συνήθως μία έκφραση σχέσης.
  - Αν είναι ψευδής, τότε ο βρόχος **for** τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της for εντολής.
  - Αν είναι αληθής, τότε εκτελείται η ομάδα των εντολών που ονομάζεται και «σώμα του βρόχου».
- Εκτελείται η τελική_έκφραση.
  - Συνήθως, η τελική_έκφραση αλλάζει την τιμή κάποιας μεταβλητής που χρησιμοποιείται στη συνθήκη.

- Επαναλαμβάνονται συνεχώς τα βήματα (2) και (3), μέχρι η τιμή της συνθήκης να γίνει ψευδής.

Στο παρακάτω παράδειγμα η επανάληψη του for ξεκινά την πρώτη φορά με a=0 εκτυπώνει το a δηλαδή 0 και πάει πάλι από την αρχή αυξάνει κατά 1 το a, ελέγχει αν είναι μικρότερο του 5 και επαναλαμβάνει μέχρι το a να γίνει 5 και να σταματήσει η επανάληψη του for.

Παράδειγμα

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

int main()
{
    int a;
    for(a = 0; a < 5; a++)
    {
        printf("%d\n", a);
    }
    return 0;
}
```

Έξοδος:

```
0
1
2
3
4

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Η εντολή **for** συνήθως χρησιμοποιείται όταν γνωρίζουμε τον αριθμό επαναλήψεων που πρέπει να εκτελεστούν.

Παράδειγμα

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

int main()
{
    int a;
    for(a = 12; a > 2; a-=5)
        printf("%d", a);
    return 0;
}
```

Έξοδος:

```
127
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Παράδειγμα

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

int main()
{
    int a = 20, b = 5;
    for(a = 0; a + b == 5; b++)
    {
        printf("One\n");
        a = 4;
        b = 1;
    }
    printf("Val1 = %d Val2 = %d\n", a, b);
    return 0;
}
```

Έξοδος:

```
One
Val1 = 4 Val2 = 2

Process returned 0 (0x0)   execution time : -0.000 s
Press any key to continue.
```

Σε κάποια προγράμματα μπορεί να χρειαστεί να χρησιμοποιηθεί ένας επαναληπτικός βρόχος **for** στο εσωτερικό ενός άλλου **for**.

```
for(αρχική_έκφραση_1; συνθήκη_1; τελική_έκφραση_1)
{
    for(αρχική_έκφραση_2; συνθήκη_2; τελική_έκφραση_2;)
    {
        ομάδα εντολών που εκτελείται όσο η συνθήκη_2 παραμένει αληθής
     }
    ομάδα εντολών που εκτελείται όσο η συνθήκη_1 παραμένει αληθής
}
```

Παράδειγμα

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

int main()
{
    int a, b;
    for(a = 0; a < 2; a++)
    {
        printf("One\n");
        for(b = 0; b<2; b+=2)
            printf("Two ");
    }
    return 0;
}
```

Έξοδος:

```
One
Two One
Two
Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

6.2 Βρόγχος **while**

Η εντολή **while**, όπως και η εντολή **for**, χρησιμοποιείται για τη δημιουργία επαναληπτικών βρόχων.

Γενική σύνταξη:

```
while(συνθήκη)
{
ομάδα εντολών που εκτελείται όσο η συνθήκη είναι αληθής
}
```

Εκτέλεση Εντολής while

- Γίνεται έλεγχος της τιμής της συνθήκης (η οποία είναι συνήθως μία σχεσιακή έκφραση)
- Αν η συνθήκη είναι ψευδής (false) τότε ο **while** βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της **while** εντολής.
- Αν η συνθήκη είναι αληθής (true) τότε εκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα {} και η τιμή της συνθήκης ελέγχεται πάλι.
  - Αν η τιμή της συνθήκης γίνει ψευδής (false), τότε ο **while** βρόχος τερματίζεται.
  - Αν όχι, επανεκτελείται η ομάδα των εντολών του βρόχου **while**.

Η παραπάνω διαδικασία επαναλαμβάνεται μέχρι η τιμή της συνθήκης να γίνει ψευδής.

⚠ Η εντολή **while** χρησιμοποιείται συνήθως όταν δεν γνωρίζουμε τον ακριβή αριθμό των επαναλήψεων που θέλουμε να εκτελεστεί η ομάδα των εντολών μας.

Παρατηρήσεις

- Η εντολή while(x) είναι ισοδύναμη με την while(x != 0) με τον δεύτερο τρόπο μα χρησιμοποιείται πιο συχνά για να είναι πιο ευανάγνωστο το πρόγραμμα.
- Αντίστοιχα, για τον ίδιο ακριβώς λόγο προτείνεται το while(x == 0) αντί του while(!x).
- Όταν σε μία while εντολή η συνθήκη είναι πάντα αληθής, τότε αυτός ο while βρόχος ονομάζεται ατέρμονας καθώς δεν τερματίζεται ποτέ.

- Εάν η συνθήκη είναι εξ' αρχής ψευδής, τότε δεν θα εκτελεστεί ποτέ το μπλοκ εντολών της while.

Παράδειγμα

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

int main()
{
    int i, pos, neg;
    pos = neg = 0;
    while(1)
    {
        printf("Enter number: ");
        scanf("%d", &i);

        if(i == 0)
            break;
        else if(i >0)
            pos++;
            else
            neg++;
    }
    printf("Pos = %d Neg = %d\n", pos, neg);
    return 0;
}
```

Έξοδος:

```
Enter number: 1
Enter number: 20
Enter number: 35
Enter number: 60
Enter number: -3
Enter number: -7
Enter number: 9
Enter number: 0
Pos = 5 Neg = 2

Process returned 0 (0x0)   execution time : 10.915 s
Press any key to continue.
```

6.3 ΒΡΟΧΟΣ do-while

Χρησιμοποιείται για τη δημιουργία επαναληπτικών βρόχων που εκτελούνται τουλάχιστον μία φορά και η γενική της σύνταξη είναι:

```
do {
  ομάδα εντολών που εκτελείται κατ' επανάληψη όσο η συνθήκη είναι αληθής
   } while (συνθήκη);
```

Βήματα Εκτέλεσης **do-while**

- Εκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα {}.
- Γίνεται έλεγχος της τιμής της συνθήκης (η οποία είναι συνήθως μία λογική έκφραση σύγκρισης).
  - Αν η συνθήκη είναι ψευδής (false) ο do-while βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της do-while.
  - Αν η συνθήκη είναι αληθής (true) τότε επανεκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα {}.
  - Το βήμα αυτό επαναλαμβάνεται μέχρι η τιμή της συνθήκης να γίνει ψευδής.

Κάθε **do-while** βρόχος εκτελείται τουλάχιστον μία φορά, ακόμα κι αν η συνθήκη του βρόχου είναι ψευδής.

Παράδειγμα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main()
5   {
6       int i = 5;
7       do
8       {
9           printf("text\n");
10          i += 5;
11      }while(i >10);
12      return 0;
13  }
14
```

Έξοδος:

```
text

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

6.4 Η Εντολή **break**

Χρησιμοποιείται για τον άμεσο τερματισμό ενός επαναληπτικού βρόχου (π.χ. for, while ή do-while) ή σετ εντολών switch-case. Στους επαναληπτικούς βρόχους, μετά την εκτέλεση της εντολής **break** το πρόγραμμα συνεχίζει με την εκτέλεση της πρώτης εντολής μετά τον βρόχο. Ωστόσο, η εκτέλεση της εντολής **break** μέσα σε έναν ένθετο επαναληπτικό βρόχο προκαλεί τον τερματισμό μόνο του βρόχου στον οποίο η ίδια περιέχεται.

Παράδειγμα

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

int main()
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            break;
        printf("%d", i);
    }
    printf("\ni = %d\n", i);
    return 0;
}
```

Έξοδος:

```
1234
i = 5

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

6.5 Η Εντολή **continue**

Χρησιμοποιείται μέσα σε έναν επαναληπτικό βρόχο (π.χ. for, while ή do-while) και προκαλεί την άμεση διακοπή της εκτέλεσης της ομάδας των εντολών της τρέχουσας επανάληψης και την έναρξη της επόμενης επανάληψης. Συνεπώς, οι εντολές ανάμεσα στην εντολή **continue** και στο τέλος του βρόχου δεν εκτελούνται για την τρέχουσα επανάληψη.

Παράδειγμα

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

int main()
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            continue;
        printf("%d", i);
    }
    printf("\ni = %d\n", i);
    return 0;
}
```

Έξοδος:

```
1234678910
i = 11

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

7. Πίνακες

Είναι δομές δεδομένων που αποτελούνται από στοιχεία ίδιου τύπου (π.χ. πίνακας ακεραίων, πραγματικών, χαρακτήρων κλπ.) και διακρίνονται σε πίνακες μιας διάστασης και πολλών διαστάσεων. Το πρώτο στοιχείο του πίνακα είναι στη [0]

Μονοδιάστατοι Πίνακες

Όπως και με τις μεταβλητές ένας πίνακας πρέπει να οριστεί πριν χρησιμοποιηθεί. Για τον ορισμό του πίνακα χρειάζεται το **όνομα** του πίνακα, ο **τύπος δεδομένων** που θα αποθηκεύσει και το **πλήθος των στοιχείων** που περιέχει ο πίνακας.

τύπος_δεδομένων όνομα_πίνακα [πλήθος_στοιχείων_πίνακα]

Π.χ. **int** grades [10];

Για τη δήλωση του μεγέθους ενός πίνακα χρησιμοποιείται πολύ συχνά η οδηγία #define
π.χ. #define SIZE 10
int grades[SIZE];

Παράδειγμα

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

int main()
{
    int a[3] = {1,2,3};
    int b[4] = {40,50,60,70};
    b[a[2]] = 10;
    printf("%d %d %d\n", b[0], b[1], b[2]);
    return 0;
}
```

Έξοδος:

```
40 50 60

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Δισδιάστατοι Πίνακες

Αποτελούνται από γραμμές και στήλες και η γενική περίπτωση ορισμού τους έχει ως εξής:

τύπος_δεδομένων όνομα_πίνακα [πληθος_γραμμών] [πλήθος_στηλών]

Παράδειγμα

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

int main()
{
    int i, j, arr[50][100];
    for(i = 0; i < 50; i++)
    {
        for(j = 0; j < 100; j++)
        {
            arr[i][j] = 300;
            printf("arr[%d][%d] = %d\n", i, j, arr[i][j]);
        }
    }
    return 0;
}
```

Στο παραπάνω παράδειγμα ορίζεται ένας δυσδιάστατος πίνακας με 50 γραμμές και 100 στήλες που γεμίζει με τον αριθμό 300 όλα του τα στοιχεία.

Αλφαριθμιτικά

Ένα αλφαριθμητικό (string) είναι μία ακολουθία χαρακτήρων, ένας πίνακας χαρακτήρων, που πρέπει να τελειώνει με έναν ειδικό χαρακτήρα, τον '\0'. Ο χαρακτήρας αυτός, ονομάζεται τερματικός χαρακτήρας (null character) και θέτει το τέλος του αλφαριθμητικού.

π.χ. **char** msg[10] που δηλώνει ένα πίνακα με δέκα στοιχεία. Ο msg μπορεί να δεχτεί μέχρι 9 χαρακτήρες κι όχι δέκα καθώς η τελευταία θέση δεσμεύεται από τον τερματικό χαρακτήρα ‘\0’.

Η αρχικοποίηση των αλφαριθμητικών (strings) γίνεται στη δήλωση τους π.χ.

char msg[] = “Hello”; ή

char msg[]={‘H’, ’e’, ’l’, ’l’, ’o’, ’\0’}; ή

char msg[10]= “Hello”;

Στο πρόγραμμα μετά τις δηλώσεις οι αλλαγές του string δεν γίνονται με ανάθεση όπως θα περίμενε κανείς π.χ. η εντολή msg= “Good bye” ή η msg={‘H’, ‘i’, ‘\0’}’δεν είναι ορθές. Οι επεξεργασία των strings γίνεται με συναρτήσεις βιβλιοθήκης όπως:

- strlen(s), υπολογίζει το μέγεθος του string. Προσοχή δεν είναι ο συνολικός διαθέσιμος χώρος που έχει δεσμευτεί, αλλά το πόσοι χαρακτήρες υπάρχουν πριν το ‘\0’
  π.χ. char msg[10]=“Hello”;
  printf(“%d”, strlen(msg));
  Εκτυπώνει: 5.
  Δηλαδή τον αριθμό των χαρακτήρων έως το \0
  Προσοχή: Το strlen δεν μας λέει το μέγιστο μέγεθος του string. Αυτό είναι 10 χαρακτήρες και το ξέρουμε από την αρχική δήλωση πριν το compile

0	1	2	3	4	5	6	7	8	9
H	e	l	l	o	\0				?

- strcpy(s1,s2), αντιγράφει το s2 στο s1
- strcat(s1,s2), προσθέτει το s2 στο s1.
- strcmp(s1,s2), συγκρίνει το s1 με s2 και επιστρέφει θετική τιμή εάν s1 μεγαλύτερο (αλφαβητικά) από το s2, μηδέν αν είναι ίσα, και αρνητική τιμή εάν s1 μικρότερο από s2. (Η σύγκριση γίνεται βάση του πίνακα ASCII)

Αρχικοποίηση Πινάκων

```
int b[] = {3,12,6}
```

Στον ορισμό του πίνακα δεν έχουμε βάλει μέγεθος αλλά επειδή τα στοιχεία που δίνουμε είναι 3 ο πίνακας έχει μέγεθος 3 με b[0]=3, b[1]=12, b[2]=6

```
int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Στον πίνακα days έχουμε αρχικοποιήσει τα 12 στοιχεία του με τους αριθμούς μέσα στις αγκύλες χωρισμένους με κόμμα (,).

Το ίδιο είναι αν δεν ορίσουμε το μέγεθος του πίνακα και απλά βάλουμε τα στοιχεία του μέσα στις αγκύλες

```
int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Για τη δήλωση πίνακα πέντε χαρακτήρων χρησιμοποιούμε `char table[5];`

Για την αλλαγή της τιμής της θέσης 0 του πίνακα χρησιμοποιούμε `table[0]=’a’;`

Για την αλλαγή τιμών του πίνακα με τιμές εισόδου από το πληκτρολόγιο και εκτύπωση τιμών του πίνακα χρησιμοποιούμε

```
scanf("%c %c", &table[0], &table[1]);
printf("Table %c %c\n", table[0], table[1]);
```

Για τον ορισμό δισδιάστατου πίνακα 2 γραμμών και 3 στηλών χρησιμοποιούμε
με στοιχεία 1 2 3
4 5 6

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

Το ίδιο είναι και αν χρησιμοποιήσουμε:

```
int grades[2][3]={1,2,3,4,5,6} ;    ή
int grades[2][3]={{1,2,3},{4,5,6}};
```

8. Συναρτήσεις

Οι συναρτήσεις είναι ανεξάρτητα τμήματα κώδικα που εκτελούν μια ομάδα εντολών όταν κληθούν και προαιρετικά επιστρέφουν μια τιμή. Οι συναντήσεις βοηθούν στην καλύτερη οργάνωση του προγράμματος και θεωρούνται η βάση του δομημένου προγραμματισμού.

Δήλωση Συναρτήσεων

Για να ορίσουμε μία συνάρτηση δηλώνουμε τον **τύπο δεδομένων** που θα επιστρέφει, μετά το **όνομά** της και μέσα σε παρενθέσεις τις **παραμέτρους** που δέχεται ορίζοντας τύπο και όνομα για κάθε μία.

τύπος_επιστροφής όνομα_συνάρτησης(**τύπος_παραμ_1** όνομα_1, **τύπος_παραμ_2** όνομα_2, …, **τύπος_παραμ_ν** όνομα_ν)

Το όνομα_συνάρτησης πρέπει να είναι μοναδικό ενώ αν η συνάρτηση προέρχεται από βιβλιοθήκη συναρτήσεων τότε η δήλωση πρέπει να περιέχεται από ξεχωριστό αρχείο και να συμπεριλαμβάνεται στο πρόγραμμα με την εντολή

#include <stdio.h>.

Ειδάλλως, αν η δήλωση της συνάρτησης δεν εμπεριέχεται σε ξεχωριστό αρχείο, δηλώνεται πριν από τη συνάρτηση **main()**.

Παρατηρήσεις

- Μια συνάρτηση δεν μπορεί να επιστρέψει πίνακα, να είναι δηλαδή τύπου πίνακα.
- Ο **τύπος_επιστροφής** μπορεί να είναι οποιοσδήποτε τύπος δεδομένων της C, όπως char, int, double, δείκτης σε κάποιον τύπο, … κτλ. Ο τύπος επιστροφής **void** που θα δούμε παρακάτω χρησιμοποιείται όταν δεν επιστρέφεται τιμή.
- Η εντολή **return** χρησιμοποιείται για άμεσο τερματισμό της συνάρτησης.
- Ο **τύπος επιστροφής** της συνάρτησης και ο **τύπος δεδομένων** της παράστασης που ακολουθεί την **return** θα πρέπει να είναι ο ίδιος.
- Με την εντολή **return** επιστρέφεται μόνο μία τιμή
- Εάν δεν υπάρχει συμφωνία είτε στο πλήθος είτε στον τύπο των παραμέτρων, τότε ενημερώνει τον προγραμματιστή με ένα λάθος μεταγλώττισης.

8.1 Είδη Συνάρτησης

Όταν καλείται μία συνάρτηση, η εκτέλεση του προγράμματος συνεχίζει με την εκτέλεση του κώδικα της συνάρτησης και επιστρέφει στην επόμενη εντολή από το σημείο της κλήσης.

Συνάρτηση Χωρίς Παραμέτρους

Η κλήση μία συνάρτησης που δεν δέχεται παραμέτρους σημαίνει ότι δεν της μεταβιβάζεται κάποια πληροφορία. Έτσι, η κλήση της γίνεται γράφοντας (μέσα στο πρόγραμμα, στο σημείο που επιθυμούμε να την καλέσουμε) το όνομά της, ακολουθούμενη από κενές παρενθέσεις) ή γράφοντας τη λέξη **void** μέσα στις παρενθέσεις στην περίπτωση που δεν επιστρέφει τιμές.

Στο παρακάτω παράδειγμα έχουμε μία συνάρτηση που όταν κληθεί τυπώνει 20 φορές μία τελεία και δεν επιστρέφει τίποτα.

Παράδειγμα

```
void typedots()
{
        for (int i=0; i<20; i++) printf(“.”);
}
```

Συνάρτηση Με Παραμέτρους

Η κλήση μίας συνάρτησης που δέχεται παραμέτρους σημαίνει ότι στη συνάρτηση μεταβιβάζεται πληροφορία μέσω των ορισμάτων της. Το πλήθος των ορισμάτων και οι τύποι τους πρέπει να ταιριάζουν με τη δήλωση της συνάρτησης. Όταν καλείται μία συνάρτηση οι τιμές της λίστας των παραμέτρων εκχωρούνται μία-προς-μία στις παραμέτρους της συνάρτησης.

Αν στο προηγούμενο παράδειγμα της συνάρτησης typedots() θέλαμε να είναι παράμετρος που περνά στην συνάρτηση το πόσες τελείες θα τυπώσει τότε θα γράφαμε τη συνάρτηση με παραμέτρους όπως παρακάτω.

Παράδειγμα

```
void typedots(int n)
{
        for (int i=0; i<n; i++) printf(“.”);
}
```

8.2 Μεταβίβαση Τιμών σε Συνάρτηση

Υπάρχουν δύο τρόποι μεταβίβασης τιμών σε μια συνάρτηση. Η κλήση συνάρτησης μέσω τιμής και η κλήση μέσω αναφοράς.

Κλήση Μέσω Τιμής (call by value)

Σε αυτήν την περίπτωση έστω πως ο χρήστης στο κυρίως πρόγραμμα (βλέπε πρόγραμμα παρακάτω στο παράδειγμα) δίνει την τιμή 20 που αποθηκεύεται στη μεταβλητή i.

Την μεταβλητή αυτή την περνάμε ως όρισμα καλώντας την συνάρτηση function(i).

Το πρόγραμμα περνά στην εκτέλεση της συνάρτησης που έχει όρισμα την μεταβλητή int a.

Έτσι δημιουργείται η μεταβλητή a στη μνήμη της αποδίδεται η τιμή που ήταν αποθηκευμένη στην μεταβλητή i και ξεκινά η εκτέλεση της συνάρτησης χρησιμοποιώντας την τοπική μεταβλητή a.

Κάθε αλλαγή στην a παραμένει στο χώρο εκτέλεσης της συνάρτησης π.χ. a=20.

Όταν επιστρέψει η συνάρτηση τότε η a δεν υπάρχει στο κυρίως πρόγραμμα και η μεταβλητή i εξακολουθεί να έχει την τιμή που είχε πριν την κλήση της συνάρτησης. Η εκτύπωση της μεταβλητής το αποδεικνύει.

Άρα στην κλήση μέσω τιμής οι τιμές των ορισμάτων δεν αλλάζουν.

Παράδειγμα

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

void function(int a);

int main()
{
    int i = 10;
    function(i);
    printf("Val = %d\n", i);
    return 0;
}

void function(int a)
{
    a = 20;
}
```

Έξοδος:

```
Val = 10

Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.
```

Κλήση Μέσω Αναφοράς (call by reference)

Είδαμε πως όταν θέλουμε η συνάρτηση να επιστρέψει μία τιμή χρησιμοποιούμε την **return**. Αν θέλουμε να η συνάρτηση να κάνει αλλαγές και σε άλλες μεταβλητές που θα ισχύουν και μετά το τέλος της συνάρτησης τότε μεταβιβάζουμε ως ορίσματα μεταβλητές με κλήση μέσω αναφοράς. Με την μεταβίβαση ορισμάτων με αναφορά δεν δημιουργούνται τοπικά αντίγραφα των μεταβλητών στις συναρτήσεις αλλά δημιουργείται μία αναφορά στη θέση της μεταβλητής δηλαδή που βρίσκεται στη μνήμη. Με τον τρόπο αυτό αλλάζοντας τα περιεχόμενα της θέσης μνήμης αλλάζουμε τα περιεχόμενα της μεταβλητής και έξω από το συνάρτηση.

Παράδειγμα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   void fun(int a, int *b)
5   {
6        a = 10;
7       *b = 20;
8   }
9
10  int main()
11  {
12      int x = 3;
13      int y = 4;
14      fun(x, &y);
15      printf("x=%d y=%d\n", x, y);
16      return 0;
17  }
```

Θέση	Μεταβ	Περιεχ πριν fun()	Περιεχ μετά fun()
10000	x	3	3
10002	y	4	20
10004	a	3	
10006	b	10002	

Στο παράδειγμα καλούμε την συνάρτηση fun με παραμέτρους κλήσης τον ακέραιο x και τον δείκτη της μεταβλητής y (θέση μνήμης 10002). Στη συνάρτηση fun λαμβάνουμε αντίγραφο της ακέραιας μεταβλητής x στην a και την ακέραια θέση της μεταβλητής y στο b (περιεχόμενα της b = 10002).

Κάθε αλλαγή στην μεταβλητή a παραμένει μόνο μέσα στην εκτέλεση της συνάρτησης ενώ αλλαγές στο περιεχόμενο της θέσης που δείχνει η b είναι σαν να γίνονται και στην y. Η κλήση της a είναι με τιμή και η κλήση της b είναι με αναφορά.

Έξοδος:

```
x=3 y=20

Process returned 0 (0x0)   execution time : 0.017 s
Press any key to continue.
```

8.3 Εμβέλεια Μεταβλητών

Εμβέλεια μεταβλητής καλείται το τμήμα του προγράμματος στο οποίο η μεταβλητή είναι προσβάσιμη και εξαρτάται από το σημείο δήλωσης στο πρόγραμμα.

Τοπικές Μεταβλητές (local variables)

Δηλώνονται στο σώμα μιας συνάρτησης και μπορεί να χρησιμοποιηθούν μόνο σε αυτή τη συνάρτηση. Οι άλλες συναρτήσεις δεν βλέπουν και δεν μπορούν να τροποποιήσουν αυτήν την μεταβλητή. Έτσι, μπορούμε να δηλώσουμε μεταβλητή με το ίδιο όνομα και σε άλλες συναρτήσεις. Συνήθως αυτό γίνεται σε μεταβλητές που είναι σε for ή προσωρινές (π.χ. for(i = 0; i < 20; i++). Τέλος, πριν χρησιμοποιηθούν στη συνάρτηση πρέπει να αρχικοποιούνται. Επίσης αν μία μεταβλητή οριστεί μέσα σε ένα τμήμα κώδικα π.χ. for(int i = 0; i < 20; i++){ εντολές} τότε μπορεί να χρησιμοποιηθεί μόνο μέσα στο τμήμα αυτού του κώδικα.

Παράδειγμα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   void test();
4
5   int main()
6
7   {
8       int i = 10;
9       test();
10      printf("I_main = %d\n", i);
11      return 0;
12  }
13  void test()
14  {
15      int i;
16      i = 200;
17      printf("I_test = %d\n", i);
18  }
19
```

Έξοδος:

```
I_test = 200
I_main = 10

Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Στατικές Μεταβλητές

Αποκτούν αρχική τιμή μόνο την πρώτη φορά που καλείται η συνάρτηση, ενώ στις επόμενες κλήσεις οι μεταβλητές διατηρούν την τελευταία τους τιμή και δεν αρχικοποιούνται πάλι.

π.χ.

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

void test();

int main()
{
    test();
    test();
    test();
    return 0;
}
void test()
{
    static int i = 100;
    int j = 0;
    i++;
    j++;
    printf("%d %d\n", i, j);
}
```

Έξοδος:

```
101 1
102 1
103 1

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Καθολικές Μεταβλητές

Δηλώνονται έξω από οποιαδήποτε συνάρτηση. Η εμβέλειά τους εκτείνεται από το σημείο δήλωσής της μέχρι το τέλος του αρχείου στο οποίο δηλώνονται. Όλες λοιπόν οι συναρτήσεις που ορίζονται μετά από το σημείο δήλωσης της καθολικής μεταβλητής έχουν πρόσβαση και μπορούν να τροποποιούν την τιμή της.

Παράδειγμα

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

void add();
void sub();

int glob = 10;

int main()
{
    add();
    printf("Val = %d\n", glob);
    sub();
    printf("Val = %d\n", glob);
    return 0;
}

void add()
{
    glob++;
}

void sub()
{
    glob--;
}
```

Έξοδος:

```
Val = 11
Val = 10

Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```

Εξωτερικές Καθολικές Μεταβλητές

Μεταβλητές που πρέπει να είναι ορατές σε περισσότερα από ένα αρχεία και δηλώνονται σαν καθολικές στο αρχείο που χρησιμοποιείται περισσότερο.

```
extern int size;      /* ενημερώνει ότι η ακέραια μεταβλητή size δε
                         δηλώνεται σε αυτό το αρχείο του προγράμματος αλλά
                         σε κάποιο άλλο*/
```

8.4 Συναρτήσεις με Παράμετρο Πίνακα

```
void test(int arr[]); /*δήλωση συνάρτησης με παράμετρο πίνακα και δεν επιστρέφει τίποτα */
```

Παράδειγμα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   void function(int arr[]); /* Δήλωση συνάρτησης που έχει σαν παράμετρο έναν πίνακα ακεραίων και δεν επιστρέφει τίποτα. */
5
6   int main()
7
8   {
9       int pin[100];
10      function(pin);   /* Στην κλήση της συνάρτησης γράφουμε το όνομα του πίνακα χωρίς τις αγκύλες*/
11      return 0;
12  }
13
14  void function(int arr[])
15  {
16      /* Σώμα συνάρτησης. */
17  }
18
```

Εάν έχουμε δηλώσει τον πίνακα arr, ισχύει ότι:

arr == &arr[0] και γενικά **arr + n == &arr[n]**

Άρα, όταν μια συνάρτηση δέχεται σαν παράμετρο το όνομα ενός πίνακα, τότε στη συνάρτηση μεταβιβάζεται η διεύθυνση του πρώτου στοιχείου κι όχι αντίγραφο του πίνακα.

Συνήθως, οι συναρτήσεις που δέχονται πίνακα σαν παράμετρο, δέχονται άλλη μια συνάρτηση η οποία δηλώνει το μέγεθος του πίνακα. Η δήλωση γράφεται είτε

void function(int *arr); είτε **void** function(int arr[]);

Η συνάρτηση function() δεν μπορεί να μεταβάλλει τις τιμές των στοιχείων ενός πίνακα arr αν δηλωθεί ως σταθερή (const).

void function(**const** int arr[]);

π.χ.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   void test(int arr[]);
5
6   int main()
7
8   {
9       int i, array[5] = {10, 20, 30, 40, 50};
10
11      test(array);
12
13      for(i = 0; i < 5; i++)
14          printf("%d ", array[i]);
15      return 0;
16
17  }
18
19  void test(int arr[])
20  {
21      arr[0] = arr[1] = 0;
22  }
23
```

Έξοδος:

```
0 0 30 40 50
Process returned 0 (0x0)   execution time : -0.000 s
Press any key to continue.
```

9. Δομές

Δομή είναι συλλογή από μεταβλητές οποιουδήποτε τύπου, οι οποίες συνήθως χρησιμοποιούνται με σκοπό να ομαδοποιηθούν πληροφορίες που περιγράφουν μια λογική οντότητα. Με τις δομές μπορούμε να περιγράψουμε πληροφορίες της καθημερινότητας (π.χ. στοιχεία μαθητή).

Το πρότυπο δομής καθορίζει τον τύπο και το πλήθος των μεταβλητών που περιέχει μια δομή.

```
struct όνομα_προτύπου
{
        τύπος_πεδίου_1  όνομα_πεδίου_1;
        τύπος_πεδίου_2  όνομα_πεδίου_2;
                ……
        τύπος_πεδίου_ν  όνομα_πεδίου_ν;
}
```

Δήλωση Δομής

Η δήλωση δομής μπορεί να γίνει με δύο τρόπους:

A. Θεωρώντας δηλωμένο το πρότυπο.
   π.χ. **struct** person person_1;

B. Ταυτόχρονα με τη δήλωση προτύπου.
   π.χ. **struct** book
   {
   char title[100];
   int year;
   float price;
   } book_1, book_2

Παράδειγμα

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

struct date
{
    int day;
    int month;
    int year;
};

int main()
{
    struct date date_1;
    printf("%d\n", sizeof(date_1));
    return 0;
}
```

Έξοδος:

```
12

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Η δήλωση int του προγράμματος σε Intel PC με GCC compiler έχει μέγεθος 4 bytes.

Άρα το μέγεθος της δομής date είναι 3 ακέραιοι με συνολικά 12 bytes. Στο Arduino το μέγεθος των ακεραίων είναι 2 bytes και άρα η έξοδος θα ήταν 6 bytes.

Δήλωση Δομής με Χρήση Προσδιοριστικού **typedef**

```
typedef unsigned int size_t;
```

Με αυτή τη δήλωση δημιουργείται νέος τύπος δεδομένων που ονομάζεται size_t και είναι συνώνυμος του unsigned int, δηλαδή unsigned int i; και size_t i; είναι ισοδύναμα.

Αρχικοποίηση Πεδίων Δομής

Από τη στιγμή που οριστεί η μεταβλητή τύπου δομής και δεσμευτεί μνήμη για τα μέλη της μπορούμε να ορίσουμε ή να διαβάσουμε τις τιμές αυτών. Η προσπέλαση των μελών της δομής γίνεται με τη χρήση του τελεστή τελείας (.)

Παράδειγμα

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

struct book
{
    char title[100];
    int year;
    float price;
};
int main()
{
    struct book book_1;
    strcpy(book_1.title, "Literature");
    book_1.year = 2010;
    book_1.price = 10.85;
    printf("%s %d %.2f\n", book_1.title, book_1.year, book_1.price);
    return 0;
}
```

Έξοδος:

```
Literature 2010 10.85

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Αντιγραφή-Σύγκριση Δομών

Για να γίνει αντιγραφή μιας δομής σε μία άλλη πρέπει οι δομές να είναι ίδιου τύπου.

Παράδειγμα

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

struct student
{
    int code;
    float grd;
};

int main()
{
    struct student stud1, stud2;
    stud1.code = 1234;
    stud1.grd = 6.7;
    stud2 = stud1;
    printf("Code: %d Grade: %.2f\n", stud2.code, stud2.grd);
    return 0;
}
```

Έξοδος:

```
Code: 1234 Grade: 6.70

Process returned 0 (0x0)   execution time : 0.001 s
Press any key to continue.
```

⚠ Εκτός από ανάθεση καμία άλλη ενέργεια δεν επιτρέπεται μεταξύ δομών.

Δομή με Πίνακες

Είναι συνηθισμένο να χρησιμοποιούνται πίνακες στις δομές.

Παράδειγμα

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

struct student
{
    int age;
    char name[50];
    float grades[10];
};

int main()
{
    struct student stud_1;
    strcpy(stud_1.name, "somebody");
    stud_1.grades[0] = 8.5;
    stud_1.grades[1] = 7.5;
    printf("Values: %s %c %c\n", stud_1.name, stud_1.name[0], *stud_1.name);
    return 0;
}
```

Έξοδος:

```
Values: somebody s s

Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```

Δομή που Περιέχει Δομή

Μια δομή μπορεί να περιέχει μια ή περισσότερες δομές που ονομάζονται ένθετες. Το πρότυπο ένθετης δομής πρέπει να δηλώνεται πριν τη δήλωση της δομής στην οποία περιέχεται αλλιώς ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους.

Παράδειγμα

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

struct date
{
    int day;
    int month;
    int year;
};

struct product
{
    char name[50];
    double price;
    struct date s_date;
    struct date e_date;
};

int main()
{
    struct product prod_1;
    strcpy(prod_1.name, "product");
    prod_1.s_date.day = 1;
    prod_1.s_date.month = 9;
    prod_1.s_date.year = 2012;

    prod_1.e_date.day = 1;
    prod_1.e_date.month = 9;
    prod_1.e_date.year = 2015;

    prod_1.price = 7.5;
    printf("The products's life is %d years\n", prod_1.e_date.year - prod_1.s_date.year);
    return 0;

}
```

Έξοδος:

```
The products's life is 3 years

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```

10. Βιβλιογραφία

1. Purdum Jack, "*Beginning C for Arduino*", Apress, 2012
2. McEoberts Michael, "*Beginning Arduino*", Apress, 2012, 2nd Edition
3. Bayle Julien, "*C Programming for Arduino*", PACT Publishing, 2013
4. Gautam, "*Program Arduino Uno in C Language*", Instructubles, 2016
5. Evans Brian "*Beginning Arduino Programming*", Apress, 2011
6. Evans Brian, "Arduino Programming Notebook, Creative Commons, 2007
7. Okamura Allison, "*Arduino Programming Language*", Stanford University, 2014
8. Kochan Stephen, "*Programming in C*", Sams Publishing, 2005, 3rd Edition
9. Kernighan B., Ritchie D., "*The C Programming Language*", Prentice Hall, 1988, 2nd Ed.
10. Πολυχρονόπουλος Ελευθέριος, «*Εισαγωγή στο Διαδικαστικό Προγραμματισμό C*», Τμ. Μηχ. Η/Υ & Πληροφορικής, Παν. Πατρών, 2008
11. Singh Chaitanya, "*C Tutorial – Learn C Programming with examples*", http://beginnersbook.com/2014/01/c-tutorial-for-beginners-with-examples/, 2015
12. Tutorials Point, "*Learn C Programming*", https://www.tutorialspoint.com/cprogramming/cprogramming_tutorial.pdf, 2014