Γλώσσα προγραμματισμού C

Εισαγωγή. Γιατί C;

Από τον Reinhardt Weber

Πολλοί ηλεκτρονικοί θα έχουν χρησιμοποιήσει μικροεπεξεργαστές με επιτυχία αλλά και θα έχουν γράψει προγράμματα σε assembly. Φυσικά όσο μεγαλώνει το μέγεθος και η πολυπλοκότητα ενός τέτοιου προγράμματος τόσο περισσότερο χρειαζόμαστε ένα περιβάλλον προγραμματισμού που είναι εύχρηστο και αποτελεσματικό. Όποιος έχει προσπαθήσει να φτιάξει μια μαθηματική εξίσωση στην assembly (όπως 1/x, sin(x)) θα καταλαβαίνει τι εννοώ. Μια γλώσσα προγραμματισμού όπως η C μας προσφέρει αρκετά προτερήματα στον προγραμματισμό των μικροελεγκτών, καθώς τα προγράμματα που γράφονται σε τέτοιο περιβάλλον είναι μεταβιβάσιμα. Εν'ολίγοις ο σκελετός του προγράμματος μπορεί να χρησιμοποιηθεί και από άλλους μικροελεγκτές εφόσον ορίσουμε πρώτα τις θύρες και τροιποποιήσουμε τις ρυθμίσεις των καταχωρητών των συναρτήσεων (functions). Ειδικά προγραμματιστές υποστηρίζουν πως ένα πρόγραμμα που χρειαζόταν 14 ημέρες για να κατασκευαστεί χρειάζεται μόνο 2-3 στην C. Συν τους άλλους πολλοί κατασκευαστές ολοκληρωμένων προσφέρουν δωρεάν εργαλεία για τον αποτελεσματικό προγραμματισμό στην C.

Όπως γνωρίζουμε όλοι φυσικά, "η Ρώμη δε φτιάχτηκε σε μιά ημέρα" και όπως είναι λογικό το ίδιο ισχύει και για τα προγράμματα. Για να μπορέστε να προγραμματίσεττε στην C θα χρειαστείτε μια εισαγωγή στην γλώσσα αυτή, μερικά παραδείγματα αλλά και κάποια γνώση των τεχνικών όρων. Το τελευταίο φαίνεται ιδιαίτερα σημαντικό αφού πολλοί ηλεκτρονικοί έχουν κατά καιρούς ρωτήσει "στα φόρα" τι σημαίνει και ποιά είναι η λειτουργία πολλών όρων στην C.

Στα παρακάτω κεφάλαια θα προσπαθήσουμε να σας εξηγήσουμε την λειτουργία των pointers, arrays κ.τ.λ. χωρίς όμως αυτό να σημαίνει πως δεν θα χρειαστείτε άλλα βοηθήματα.

C Basics

Η δομή ενός προγράμματος στην C

Όλα τα προγράμματα στην C αποτελούνται από διάφορα μέρη όπως σχόλια, εντολές, δηλώσεις (declaration), εκφράσεις (expressions), εκχωρήσεις (assignments) και λειτουργίες (functions). Εδώ θα βρείτε ένα τέτοιο παράδειγμα.

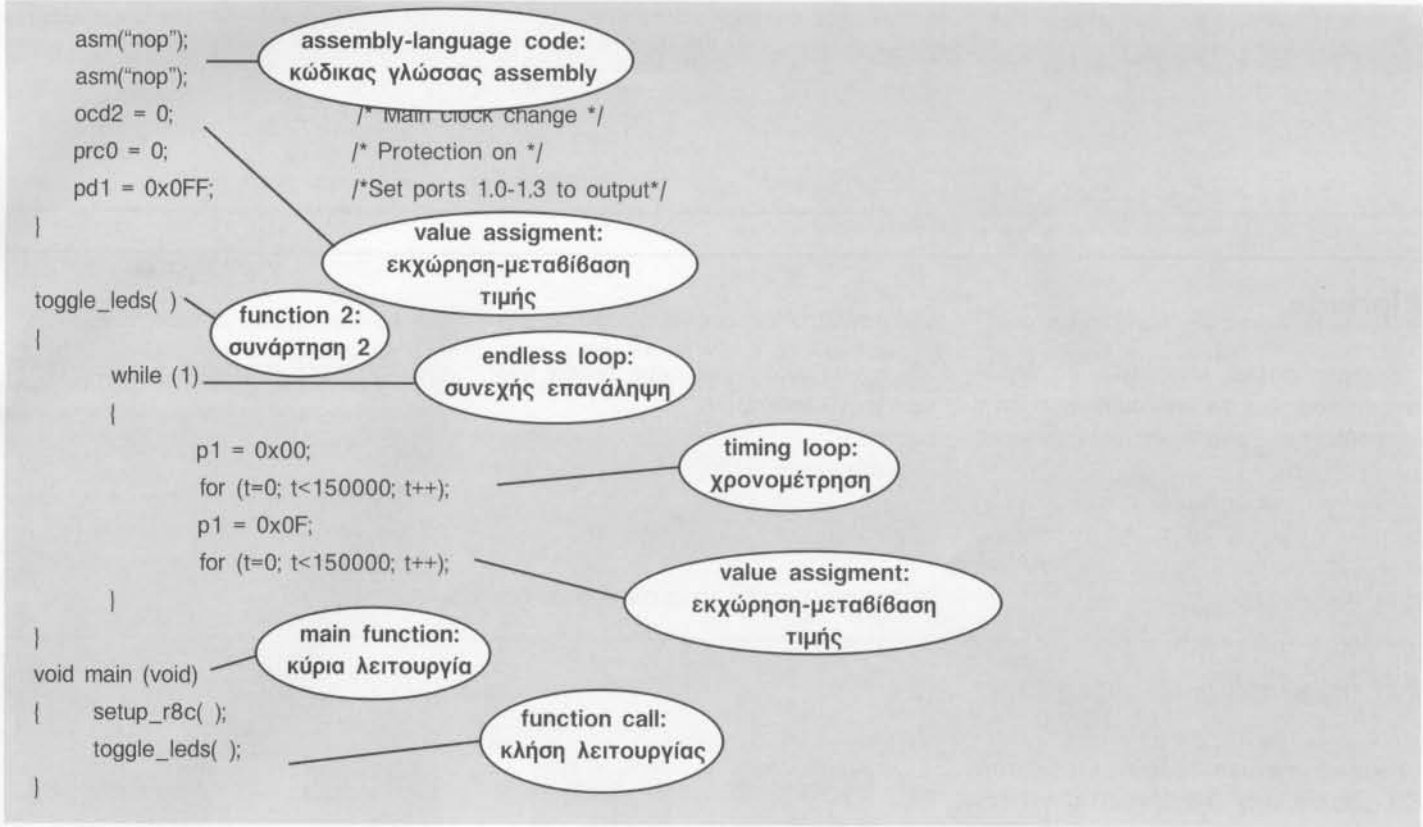
```
/*  FILE        :my1c.c                                    */
/*  DATE        :Wed, Nov 23 2005                            */
/*  DESCRIPTION :Program toggles on port_1                          */
/*  CPU TYPE    :R8C                                       */

#include "sfr_r813.h";

long t;

setup_r8c( )
{
prc0 = 1;               /* Protect off */
cm13 = 1;               /* Xin Xout */
cm15 = 1;               /* XCIN-XCOUT drive capacity: HIGH */
cm05 = 0;               /* Xin on */
cm16 = 0;               /* Main clock = No Division mode */
cm17 = 0;
cm06 = 0;               /* CM16 and CM17 enable */
asm("nop");             /* Waiting for stable of oscillation */
asm("nop");             Assembly-language code/*
```

comment: σχόλιο

compiler directive: οδηγία μεταγλώτισης

variable declaration: δήλωση μεταβλητής

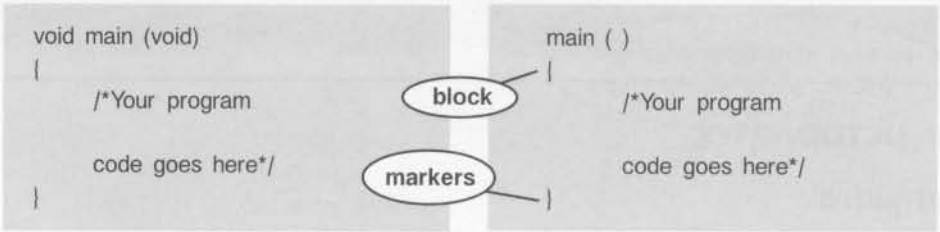
function 1: λειτουργία 1



Κύρια λειτουργία

Κάθε πρόγραμμα στη C θα πρέπει να έχει μια λειτουργία η οποία ονομάζεται κύρια λειτουργία και η οποία είναι και η πρώτη που καλείται κάθε φορά που τρέχει το πρόγραμμα. Καλά θα ήταν η κύρια ρουτίνα να αποτελείται μόνο ή κυρίως από κλήσεις λειτουργιών και όχι από τον κώδικα του προγράμματος. Μιά τέτοια δομή μας βοηθάει να καταλαβαίνουμε και να τροποποιούμε πιο εύκολα το πρόγραμμα μας.

Η κύρια λειτουργία ορίζεται όπως οι υπόλοιπες λειτουργίες.



Όλες οι εντολές και οι συναρτήσεις που ανήκουν στο main βρίσκονται ανάμεσα σε δύο αγκύλες. Οι προγραμματιστές το ονομάζουν αλλιώς και block building δηλ. δημιουργία ομάδων/πακέτων.

Στο παραπάνω παράδειγμα η λέξη void σημαίνει άδειος και δηλώνει ότι η κύρια λειτουργία δεν απαιτεί την εισαγωγή παραμέτρων αλλά ούτε και εξάγει καμμία τιμή μετά την εκτέλεση των εντολών της. Πάντως η χρήση του void δεν είναι απαραίτητη.

Σχόλια στην C

Όλες οι σειρές προγραμμάτων και φράσεις που δεν αποτελούν μέρος του προγράμματος ονομάζονται σχόλια, αγνοούνταια από τον μεταγλωτιστή και δε καταλαμβάνουν αποθηκευτικό χώρο στην μνήμη.

Σκοπός των σχολίων είναι να βοηθήσουν τόπσο εμας όσο και τους υπόλοιπους προγραμματιστές να καταλάβουμε τι κάνει σχεδόν η κάθε εντολή.

Έτσι δε θα είναι δύσκολο συντηρήσουμε και να τροποποιήσουμε το πρόγραμμα.

Σχόλια που δεν ξεπερνούν τη μιά γραμμή ξεκινάνε με δυό διαγώνιες κάθετες γραμμές και τελειώνουν με μιά διαγώνια κάθετη γραμμή.

Το ερωτηματικό (;) που χρησιμοποιούσατε στην assembly έχει διαφορετική χρήση στην C αφού δηλώνει το τέλος μιας εντολής.

```
/*
Comments are
encosed between
diagonal slashes and asterisks.*/
Τα σχόλια εισάγωνται μεταξύ
διαγωνίου και αστερίσκου
```

```
//This is a sihgle-line comment.
```

#Include

Υπάρχουν πολλές λειτουργίες και δηλώσεις οι οποίες δεν συμπεριλαμβάνονται στην C παρόλο που θα ήταν ιδιαίτερα χρήσιμες και αναγκαίες.

Πολύ συχνά θα τις βρούμε κρυμμένες μέσα σε βιβλιοθήκες και θα πρέπει να "πούμε" στον μεταγλωτιστή να τις συμπεριλάβει κατά την μεταγλώτιση.

Τα αρχεία αυτά ονομάζονται "header files" και έχουν κατάληξη .h.

Παραδείγματα:

```
#include stdio h."
```

Το παραπάνω αρχείο χρησιμοποιείται σε προγράμματα στην C που πρέπει να τρέξουν σε PC.

```
#include"sfr_r813.h"
```

Η βιβλιοθήκη renesas στην οποία έχουν ορισθεί τα ονόματα και τα ψηφία των καταχωρητών του R8C όπως p1, pd1, p1_7 κλπ.

Λέξεις κλειδιά στην C

Συνολικά υπάρχουν 32 λέξεις κλειδιά στην C οι οποίες είναι "καπαρωμένες" για τον μεταγλωτιστή.

Όλες οι λέξεις είναι γραμμένες με μικρά γράμματα και δεν επιτρέπεται να χρησιμοποιηθούν για άλλους σκοπούς.

Πολλοί μεταγλωτιστές στην C χρησιμοποιούν επιπλέον λέξεις κλειδιά έτσι ώστε να γίνεται η καλύτερη χρήση των ιδιοτήτων του μεταγλωτιστή ή μικροελεγκτή. Για τους μικροελεγκτές R8C χρησιμοποιούνται οι δίπλα:

```
auto
break
case
char
const
continue
default
do
```

```
double
else
enum
extern
float
for
goto
if
```

```
int
long
register
return
short
signed
sizeof
static
```

```
struct
switch
typedef
union
unsigned
void
volatile
while
```

```
_asm
_far
_near
```

```
asm
_Bool
far
```

```
near
restrict
inline
```

Σταθερές και μεταβλητές

Αριθμητικά συστήματα

Η γλώσσα C μπορεί να λειτουργήσει με διάφορα αριθμητικά συστήματα όπως: το δυαδικό, το δεκαδικό, το οκταδικό και το δεκαεξαδικό.

Αριθμοί χωρίς κάποιο αναγνωριστικό εκλαμβάνονται ως δεκαδικοί ενώ οι αριθμοί που ξεκινάνε με 0, 0x ή 0b εκλαμβάνονται ως οκταδικοί, δεκαεξαδικοί ή δυαδικοί αντίστοιχα.

Βάση	Σημείωση	Διαθέσιμοι χαρακτήρες	Παράδειγμα
Δεκαδικό(10)	-	0123456789	5
Οκταδικό(8)	0...	01234567	05
Δεκαεξαδικό(16)	0x	0123456789ABCDEF	0x5
Δυαδικό(2)	0b	01	0b11110000

Στη γλώσσα C χρησιμοποιείται το Αγγλικό/Αμερικάνικο σύστημα μέτρησης και επομένως η τελεία (.) λειτουργεί ως το κόμμα (,). Το κόμμα (,) χωρίζει αριθμούς. Η άνω και η κάτω τελεία (:) χαρακτηρίζει μια σειρά από αριθμούς.

Παράδειγμα:

Τύποι δεδομένων

Πριν ξεκινήσει ένα πρόγραμμα θα πρέπι να γνωρίζετε τον αποθηκευτικό χώρο που πρέπει να καταλαμβάνει μιά μεταβλητή. Βασικά πρέπει να επιλέγετε ένα τύπο ο οποίος θα κάνει όσο το δυνατόν ελάχιστη χρήση μνήμης. Τους περισσότερο σημαντικούς τύπους δεδομένων θα τους βρείτε παρακάτω:

Τύπος	Χώρος που καταλαμβάνει	Περιθώριο αριθμών
_Bool	8	0,1
char	8	0>+255
signed char	8	-128>+127
int, short	16	-32768>+32767
unsigned int	16	0>+65535
long	32	-2147483648+21474883647
float	32	-1.17..e-38F>+3.4..e-38F

Παραδείγματα:

_Bool stop_button // Η button έχει μόνο δύο καταστάσεις: on & off.

unsigned int _year // το εύρος 0>65535 επαρκεί για τον αριθμό των χρόνων.

float _volume // Αριθμοί κινητής υποδιαστολής για υπολογισμούς.

Σταθερές

Οι σταθερές είναι αριθμοί οι οποίοι δεν αλλάζουν κατά την εκτέλεση του προγράμματος.

Τέτοιοι είναι οι ακέραιοι (integers), αριθμοί κινητής υποδιαστολής (float), χαρακτήρες (char) οι οποίοι πρέπει να μπαίνουν μέσα σε quotation mark δηλ. (?).

Οι σταθερές δηλώνονται με #define λέξη κλειδί.

```
#define        <label>        value
```

Όχι γιατί είναι είναι σχετικό μόνο με τον μεταγλωτιστή.

Παραδείγματα:

```
#define true 1          // 1=αλήθεια
#define false 0         // 0=ψέμα
#define pi 3.14159      // ο παράγοντας π
#define lettter_1'A'    // ο χαρακτήρας 'A' του πληκτρολογίου.
```

Τα ονόματα των σταθερών, μεταβλητών και λειτουργιών μπορεί να είναι όποια εμείς θέλουμε αρκεί να μην περιέχουν λέξεις κλειδιά (keywords) ή σύμβολα πράξεων. Γενικά χρησιμοποιούνται μόνο λατινικοί χαρακτήρες, αριθμοί και το underscore (_). Φροντίστε να επιλέγετε ονόματα τα οποία βγάζουν κάποιο νόημα όπως alarm_btn.

Μεταβλητές

Μιά μεταβλητή είναι ένα αντικείμενο το οποίο αποθηκεύεται στην μνήμη και μπορεί να είναι αριθμοί, γράμματα ή σειρές γραμμάτων.

Στην C όλες οι μεταβλητές πρέπει να δηλωθούν πριν χρησιμοποιηθούν.

Η δήλωση θα πρέπει να τελειώνει με ένα ερωτηματικό (;).

Οι μεταβλητές ορίζονται όπως βλέπετε παρακάτω:

Το ερωτηματικό είναι απαραίτητο στο τέλος είναι απαραίτητο αφού αποτελεί μιά εντολή.

```
type        <label>        ;
```

Παραδείγματα:

```
_Bool       keypress;
long        counter;
float       radius;
```

Οι μεταβλητές έχουν τιμές χωρίς πρόσημο όπως δίπλα:

```
<label>     =     value     ;
```

Παραδείγματα:

```
keypress=1;          min_val=counterr-50;
keypress=false ;     max_val=counter*counter;
counter=100 ;        _circum=radius*2*pi;
```

Οι εντολές και οι οδηγίες που για τον επεξεργαστή αποτελουν statements (δηλώσεις) και ακολουθούνται πάντα με ερωτηματικό (;).

Πράξεις στην C

Αριθμητικές πράξεις, αριθμητικοί τελεστές

Τα αριθμητικά σύμβολα που χρησιμοποιούνται στην C είναι τα ίδια με αυτά που χρησιμοποιούμε στα κομπιουτεράκια μας.

```
+    άθροισμα        // παράδειγμα    y=x+3;
-    αφαίρεση        //               y=x-b;
*    πολλαπλασιασμός //               y=a*b;
/    διαίρεση        //               Y=a/b;
```

Το ίσον (=) εδώ πέρα δηλαδή στην γλώσσα C έχει διαφορετική σημασία από ότι στα κοινά μαθηματικά. Η μεταβλητή αριστερά του ίσον (=) παίρνει την τιμή της συνάρτησης που βρίσκεται στα δεξιά. Οι διπλανές λοιπόν εκφράσεις επιτρέπονται στην C αλλά όχι και στα "κοινά" μαθηματικά¨:

```
x=x+y;    // Υπολόγισε x+y και αποθήκευσε το αποτέλεσμα στον x.
x=-x;     // άλλαξε το πρόσημο της μεταβλητής x.
```

Σχεσιακός τελεστής

Οι σχεσιακοί τελεστές χρησιμοποιούνται για να συγκρίνονται μεταβλητές. Το αποτέλεσμα της σύγκρισης είναι true (αληθές) ή false (ψευδές).

```
>     Μεγαλύτερο από          ==  ίσο με
>=    Μεγαλύτερο από ή ίσο    !=  όχι ίσο
<     Μικρότερο από
<=    Μικρότερο από ή ίσο
```

Λογικοί τελεστές

Οι λογικοί τελεστές AND (και), OR (ή) και NOT (όχι) χρησιμοποιούνται για την εκτέλεση πράξεων της ψηφιακής λογικής.

		AND	OR	NOT
b	a	a&&b	a ll b	!a
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	1	0

Παράδειγμα:

```
if_price<=max_price&&_account>1000)
      _buy();

/*Η λειτουργία _buy() θα κληθεί εάν η if_price είναι μικρότερη ή ίση με max_price και η _account περιέχει περισσότερα από 1000 pound*/
```

Ο μεταγλωτιστής της Renesas για τον R8C περιέχει αρκετούς λογικούς τελεστές που μπορούν να χρησιμοποιηθούν σε πράξεις με ψηφία:

```
&    Για διαχείριση ψηφίων λειτουργίας AND      b  a  a·b
                                                0  0   0
|    Για διαχείριση ψηφίων λειτουργίας OR       0  1   0
                                                1  0   0
·    Για διαχείριση ψηφίων λειτουργίας XOR      1  1   1
```

Παράδειγμα:

a=10011010	a=10011010	a=10011010
b=11000011	b=11000011	b=11000011
a&b=10000010	alb=11011011	a·b=01011001

Συντομεύσεις

Προκειμένου να αποφύγετε την πληκτρολόγηση ολόκληρης της συνάρτησης στη C, έχουν οριστεί συντομεύσεις που επιταχύνουν τη διαδικασία προγραμματισμού.

Παράδειγμα:

```
for(t=0, t<100000, t++);        /*timer loop*/
```

Συντόμευση	Κανονική	Συντόμευση	Κανονική
a*=b	a=a*b	a<<=b	a=a<<b
a/=b	a=a/b	a>>=b	a=a>>b
a+=b	a=a+b	a&=b	a=a&b
a-=b	a=a-b	al=b	a=alb
a%=b	a=a%b	a·=b	a=a·b
a++	a=a+1(αύξηση)		
a—	a=a-1(μείωση)		

Λειτουργίες στην C

Το αποτέλεσμα της λειτουργίας

Οι λειτουργίες αποτελούν την βάση της C και μπορούν να κληθούν από οποιοδήποτε σημείο της κύριας ρουτίνας άλλης λειτουργίας. Κάθε πρόγραμμα πρέπει οπωσδήποτε να έχει μιά λειτουργία με την ονομασία main (κύρια) και η οποία θα καλείται με την εκκίνηση του προγράμματος. Οι λειτουργίες είναι αυτόνομα μέρη προγράμματος που εκτελούν συγκεκριμένες εργασίες όπως αυτές που υπάρχουν στα κομπιουτεράκια.

Παραδείγματος χάρη το **CE** αδειάζει τη μνήμη.

Το **1/χ** που απαιτεί μιά τιμή στο κομπιουτεράκι μας στη C θα πρέπει να δηλωθεί ώς μιά λειτουργία με μιά μεταβλητή εισόδου.

Τα +, -, *, / σε αντίθεση απαιτούν δυό παραμέτρους και στην Σ που είναι το άθροισμα απαιτούνται διάφορες παράμετροι. Οι Sin και log απαιτούν μιά παράμετρο.

Δήλωση μιάς λειτουργίας

Ο γενικός σκελετός μιας λειτουργίας στην C έχει την μορφή:

```
; Type function_name(type var1, type var2,type var3,...)        ;
```

return type - ο τύπος της παραμέτρου που θα επιστραφεί μετά την εκτέλεση της λειτουργίας

input parameter with type declaration - ο τύπος της παραμέτρου εισόδου

Παραδείγματα

Μια λειτουργία χωρίς παραμέτρους εισόδου και εξόδου όπως φαίνεται δίπλα.

```
void wait_1(void)
    asm("nop");          // Κλήση χωρίς λειτουργία στην γλώσσα assembly
```

Όπως είπαμε και προηγουμένως η λέξη void δηλώνει πως η λειτουργία wait_1 δεν απαιτεί παραμέτρους εισόδου και εξόδου και δε επιστρέφει αποτέλεσμα.

```
wait_1()
 asm("nop")
```

;

Αν μιά λειτουργία περιέχει πάνω από μία εντολή τότε αυτές θα πρέπει να ομαδοποιηθούν σε ένα πακέτο που αποτελείται από αγκύλες ({}) και χωρίς ερωτηματικό στο τέλος.

όχι;

```
wait_2()
{
    asm("nop")          //περίμενε τρεις φορές
    asm("nop")
    asm("nop")
}
```

όχι;

Κλήση μιας λειτουργίας

Οι λειτουργίες μπορούν να κληθούν πληκτρολογώντας μόνο τα ονόματα τους και αυτό μπορεί να γίνει σε οποιοδήποτε σημείο του προγράμματος.

Αφού έχει εκτελεσθεί η λειτουργία (την οποία θα αναγνωρίσετε από το ερωτηματικό στο τέλος (;)) αν αποτελείται από μία μόνο εντολή ή από τις αγκύλες ({}) αν αποτελείται από περισσότερες εντολές.

Η λέξη κλειδί return έχει διαφορετική σημασία στην C από ότι στην assembly.
Συγκεκριμένα το return δηλώνει την επιστροφή μιας τιμής και όχι το τέλος μιάς ρουτίνας.

Στην C επιτρέπεται και το nesting (φώλιασμα) δηλ. το να μπορεί μιά λειτουργία να καλεί μιά δεύτερη η οποία με την σειρά της θα καλέι μια τρίτη κ.ο.κ.

Παραδείγματα

Κλήση μιας ρουτίνας χωρίς παραμέτρους εισόδου ή επιστροφής:

```
void main(void) {
      wait_1();
}
```

Κλήση μιάς ρουτίνας από την κύρια με παραμέτρους εισόδου αλλά χωρίς παραμέτρους επιστροφής:

```
void main(void); {
      wait_3(100);
}
```

//Η σταθερά 100 μεταβιβάζεται στην λειτουργία wait_3 . Κλήση μιας λειτουργίας από την κύρια ρουτίνα με παραμέτρους εισόδου και εξόδου:

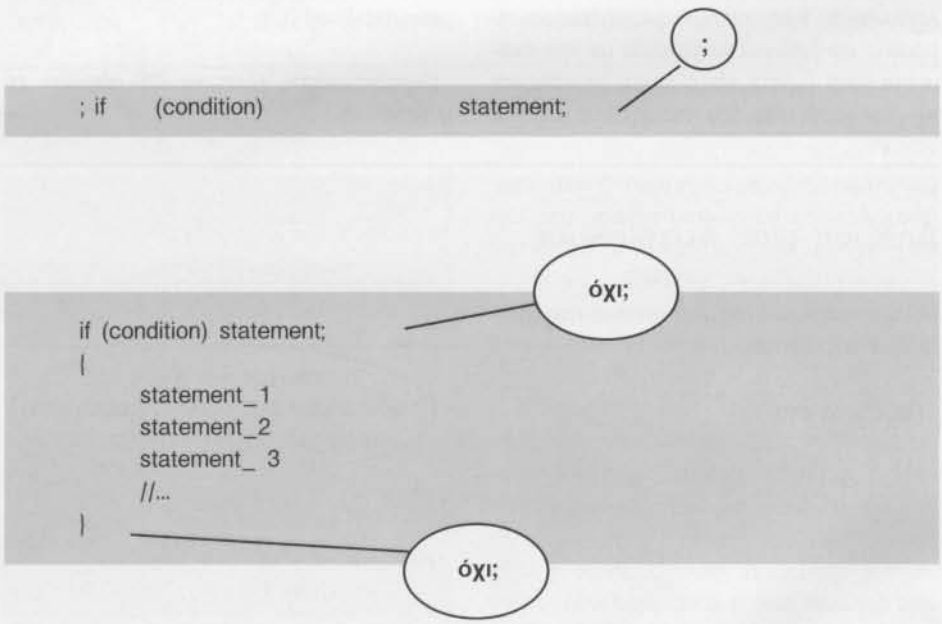
```
void main(void); {
      no_of_litres=_volume(a,b,c);
}
```

/*Οι τιμές των μεταβλητών a,b και c μεταβιβάζονται στην λειτουργία _volume.Η λειτουργία υπολογίζει το μέγεθοσ και μας επιστρέφει το αποτέλεσμα στην μεταβλητή variable no_of_litres.

Έλεγχος προγράμματος

If

Πολύ συχνά θα συναντήσετε περιπτώσεις στις οποίες μιά εντολή ή πακέτο εντολών θα πρέπει να εκτελεστεί εφόσον εκπληρωθεί πρώτα μιά προυπόθεση. Όταν εκπληρωθεί η προυπόθεση αυτή, η λειτουργία επιστρέφει την τιμή true η οποία μπορεί να είναι οποιοσδήποτε αριθμός εκτός του μηδέν. Το μηδεν ισοδυναμεί με false. Ο γενικός σκελετός είναι όπως φαίνεται δίπλα. Αν θέλουμε να εκτελεσθεί μια σειρά εντολών, τότε θα πρέπει όταν εκπληρωθεί μιά προυπόθεση να τις ομαδοποιήσουμε σε ένα πακέτο.



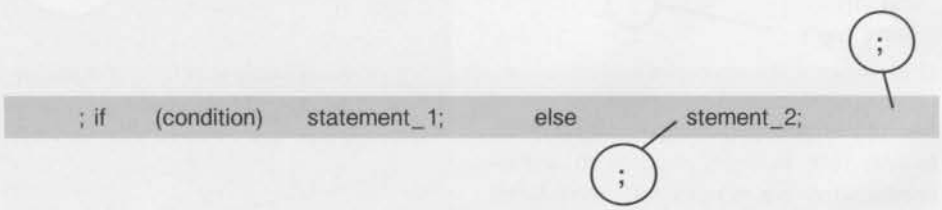
Παραδείγματα:

```
if(button==3)
      red_led=_on;
```

```
if(button==3)
{
grn_led=_off;
red_led=_on;
}
```

If...else

Χρησιμοποιούμε την τέλεση if..else όταν μιά εντολή ή πακέτο εντολών εκτελείται, όταν εκπληρώνεται μια προυπόθεση ενώ εκτελείται μιά άλλη εντολή ή πακέτο εντολών όταν αυτή δεν εκπληρώνεται.
Ο γενικός σκελετός είναι:



Αν θέλουμε να εκτελεσθεί μια σειρά εντολών, τότε πρέπει όταν εκπληρωθεί μιά προϋπόθεση, να τις ομαδοποιήσουμε σε ένα πακέτο.

Παραδείγματα:

```
if(button==3)
        red_led=_on;
else
        grn_led=_on;

if(button==3)
{
        grn_led=_off;
        red_led=_on;

} else
        grn_led_on;
```

```
if(button==3)
{
        grn_led=_off;
        red_led=_on;
}
else
{
        grn_led_on;
        red_led=off;
}
```

Switch

Αν έχουμε μια σχεσιακή τέλεση στην οποία έχουμε περισσότερα από ένα αποτελέσματα τότε αντί να χρησιμοποιήσουμε την if...else καλύτερα είναι να χρησιμοποιήσουμε την switch/case με τις πολλαπλές επιλογές. Η γενική της μορφή φαίνεται δίπλα.

Η λειτουργία (function) συγκρίνει το περιεχόμενο της μεταβλητής με την τιμή που έχει η σταθερά (constant_x) σε κάθε ξεχωριστή περίπτωση. Αν το περιεχόμενο της μεταβλητής και της σταθεράς είναι ίδιο τότε εκτελείται η αντίστοιχη εντολή (instruction_) ή πακέτο εντολών. Όταν το πρόγραμμα φτάσει σε κάποια εντολή break; τότε επιστρέφει σε αυτόν που κάλεσε την λειτουργία, την τιμή που προκύπτει από την τελευταία εντολή. Αν καμμιά από τις προυποθέσεις δεν εκπληρωθεί τότε θα εκτελεσθεί η εντολή που αντιστοιχεί στο default.

```
switch              (variable)
{
    case                        constant_1;
                                instruction_1;
                                break; (διακοπή)
    case                        constant_2;
                                instruction_2;
                                break;
    case                        constant_3
                                instruction_3
    case                        //...
                                break;
    default(προκαθορισμένο)     instruction_;
}
```

όχι;

Παράδειγμα:

```
switch              (_button)
{
    case            1:
                    red_led=_on;
                    break;

    case            2:
                    yel_led=_on;
                    break;

    case            3:
                    grn_led_on;

    default         blu_led=_on;
}
```

πακέτο

σηματοδότες

μπορεί να παραληφθεί

/*ενεργοποίησε το κόκκινο led αν η τιμή του κουμιού ισοδυναμεί με 1, ενεργοποίησε το κίτρινο led όταν η τιμή του κουμπιού ισοδυναμεί με 2 κ.ο.κ.

For

Το for χρησιμοποιούμε όταν μέρος του προγράμματος πρέπει να εκτελεσθεί πολλαπλές φορές. Ο γενικός σκελετός φαίνεται δίπλα.

```
for (start_value; end_condition; step_size)
    instruction_1;
```

Όταν καλείται ο βρόγχος for η (αρχική) start_value αποκτά την τιμή της προηγούμενης μέτρησης.

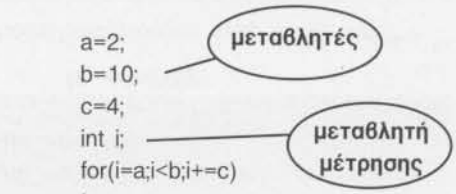
Η μεταβλητή μέτρησης αυξάνεται κατά την τιμή της step_size κάθε φορά που εκτελείται ο βρόγχος μέχρι ο έλεγχος της (τελικής) end_condition να δώσει την τιμή true. Αν θέλουμε να εκτελεσθεί μια σειρά εντολών, τότε πρέπει να βάλουμε αγκύλες ({}) για να τις ομαδοποιήσουμε σε ένα πακέτο.

Παραδείγματα:

```
int  t;
for(t=0, t<10, t++)
(αναβόσθησε) blink_led();
```

/* Η μεταβλητή t παίρνει τιμή εκκίνησης όταν μπαίνουμε στον βρόγχο for.

Στη συνέχεια καλείται η λειτουργία blink_led(); και όταν βγούμε από αυτή, η τιμή της t αυξάνει κατά μία μονάδα. Ο βρόγχος θα σταματήσει όταν η τιμή του t γίνει 9. Ο βρόγχος θα εκτελεστεί 10 φορές.



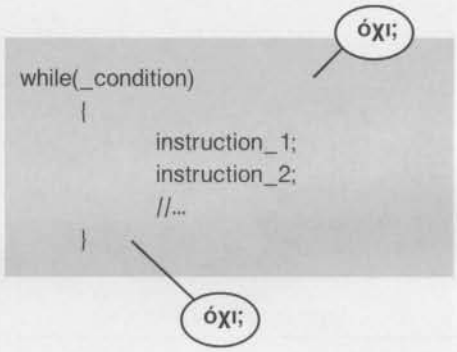

```
a=2;
b=10;
c=4;
int i;
for(i=a;i<b;i+=c)
{
red_led=_on;
red_led=_off;
}
```

/*αυτός ο βρόγχος εκτελείται μόνο δύο φορές.

While

Ένας βρόγχος while θα χρησιμοποιηθεί όταν η εκτέλεση εντολών είναι εξαρτώμενη από μιά προυπόθεση. Η γενική του μορφή φαίνεται δίπλα.

Όταν καλείται ο βρόγχος while, ελέγχεται η τιμή της προυπόθεσης και αν αυτή είναι αληθής εκτελούνται οι εντολές επανειλειμμένα μέχρι το αποτέλεσμα της προυπόθεσης να γίνει ψευδές (false).



Παράδειγμα:

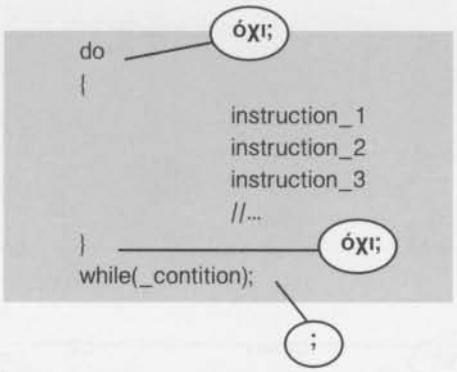

```
#define true 1
while(i=true)
{
i=button_pressed():
blink_led();
}
```

/*Η λειτουργία blink_led εκτελείται διαρκώς όσο η τιμή της i ισοδυναμεί με 1.

Do...while

Η do ...while είναι ίδια με τη while, με τη μόνη διαφορά ότι εδώ οι εντολές εκτελούνται τουλάχιστον μία φορά και στη συνέχεια γίνεται ο έλεγχος της προυπόθεσης.

Ο γενικός σκελετός φαίνεται δίπλα.



Παράδειγμα:

```
#define true 1
do
{
i=button_pressed():
blink_led();
}
whlile(i==true);
```

/*Η λειτουργία blink_led εκτελείται τουλάχιστον μία φορά*/

Appendix/παράρτημα

Header file sfr_r813.h Το αρχείο sfr_r813.h παρέχει πρόσβαση σε ειδικούς λειτουργικούς καταχωρητές του μικροελεγκτή R8C. Οι καταχωρητές αυτοί περιέχουν βασικές ρυθμίσεις για το μικροελεγκτή όπως οι κατευθύνσεις της θύρας (in/out), τις ρυθμίσεις του χρονιστή, τις ρυθμίσεις του μετατροπέα A/D, τις ρυθμίσεις του UART κ.τ.λ.π. Ο μικροελεγκτής R8C έχει περισσότερους από 50 SFR (καταχωρητές ειδικών λειτουργιών) και εδώ θα περιορίσουμε την περιγραφή μας μόνο σε μερικούς από αυτούς.

Port registers (P0, P1, P2, P3 και P4)

Η θύρα είναι ένα συγκεκριμένο κομμάτι μνήμης το οποίο επικοινωνεί με τις ακίδες του μικροελεγκτή. Κατά την εκίννηση του μικροελεγκτή δέχεται εισερχόμενα δεδομένα. Η κατεύθυνση αυτή αλλάζει χρησιμοποιώντας τον καταχωρητή port direction (PD). Τα ακόλουθα παραδείγματα δείχνουν πως αλλάζει η κατεύθυνση της θύρας:

Register Port direction pdi
i=0,1,2,3,4

b7	b6	b5	b4	b3	b2	b1	b0
d	d	d	d	d	d	d	d

d=0: input d=1:output

Παραδείγματα:

```
pd1=oxoF;
/*port1, bits 0:3=output
bits 4:7=input*/

pd2_3=1;
/* port2,bit3=output*/
```

Τα παρακάτω παραδείγματα δείχνουν το πως θα εξάγουμε δεδομένα από μιά θύρα.

/*Αν στείλουμε το 1 σε ένα καταχωρητή θύρας θα υπάρξει τάση (+5V) στην αντίστοιχη ακίδα. Αντίστοιχα ένα 0 θα δώσει στην ακίδα μηδενική τάση.

Port register pi
i=0, 1, 2, 3, 4

b7	b6	b5	b4	b3	b2	b1	b0
d	d	d	d	d	d	d	d

output d=0:gnd d=1:+Vcc

Παραδείγματα:

```
p1=0xoF;
/*port, bits 0:3=1
bits 4:7=0*/

pd2_3=0;
/*port2, bit3=0*/
```

Καταχωρητές προστασίας (PRCR)

Οι καταχωρητές αυτοί φροντίζουν να μην σβηστούν με άλλα δεδομένα άλλοι σημαντικοί καταχωρητές.

Protect Register PRCR

b7	b6	b5	b4	b3	b2	b1	b0
p	p	p	p	p	p	p	p

Write protection p=0:on p=1:off

Το ψηφίο b0 είναι το ψηφίο προστασίας-εγγραφής για το CM0, CM1, OCD, HR0, και HR1.

Παραδείγματα:

```
prc0=1;
/*write protection disabled*/

pcr0=0;
/*write protection enabled*/
```

Καταχωρητές ελέγχου ρολογιού (χρονισμού) συστήματος

Ο μικροελεγκτής R8C διαθέτει δυό ταλαντωτές που λειτουργούν ως χρονιστές της CPU.

Ο πρώτος είναι ο εσωτερικός (on chip) ταλαντωτής του και ο άλλος ο εξωτερικός ταλαντωτής.

Ο εξωτερικός ταλαντωτής (main clock) ταλαντωτής είναι συνδεμένος με τις ακίδες Xin και Xout. Οι καταχωρητές CM καθορίζουν το πως παράγεται το σήμα χρονισμού του μικροελεγκτή. Αυτό συμβαίνει επειδή μπορεί να χρησιμοποιηθεί ένας προδιαιρέτης (prescaler) για να ελλατώσουμε την συχνότητα του χρονιστή του επεξεργαστή.

Ο ταλαντωτής (Oscilator Stop Detection, OSD) χρησιμοποιείται μαζί με τους υπόλοιπυς καταχωρητές για να επιλέξουν τον χρονιστή (εσωτερικό ή εξωτερικό) και επιπλέον για να ελέγχει το σήμα του χρονιστή.

Καταχωρητής ελέγχου συστήματος χρονισμού CM1

b7	b6	b5	b4	b3	b2	b1	b0
0	0	1			1		

τοποθέτηση προδιαιρέτη

τροπος λειτουργίας εξωτερικού κρυστάλλου

κέρδος ταλαντωτή

Παραδείγματα:

```
cm13=1;
/*τα Xin/Xout στις θύρες
p46 και p47=ext.xtal*/
cm15=1;
/*τα Xin/Xout οδηγούνται σε υψηλό
δυναμικό*/
cm16=0; /*ο
προδιαιρέτης χρονισμου του CPU*/
cm17=0; /*ο
προδιαιρέτης χρονισμου του CPU*/
```

System Clock Register
CM0

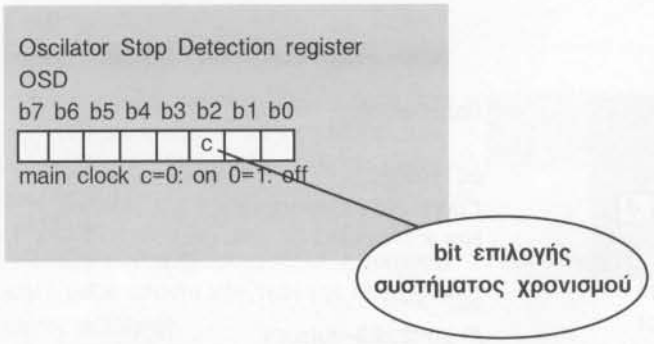
b7	b6	b5	b4	b3	b2	b1	b0
	0	0					

CM1 prescaler ομάδα bits 6 & 7

ενεργοποίηση κρυσταλλικού ταλαντωτή

Παραδείγματα:

```
cm05=0;
/*enable Xin/xout on port p4.6
and p4.7*/
cm06=0;
/*enable prescaler*/
```



Παραδείγματα:

```
osd2=0;
/*enables external xtal osc.
as CPU clock sourse*/

osd2=1;
/*disable external xtal osc.*/
```

Header file math.h

Ο φάκελλος match.h περιέχει τα περιεχόμενα της βιβλιοθήκης μαθηματικών λειτουργιών για τον R8C. Παρακάτω σας δίνουμε τις πιό σημαντικές συναρτήσεις.

Συναρτήσεις διπλού τύπου
f_name(float x);

```
sin();      sqrt();
cos();
tan();      exp();
            log();
asin();     log10();
acos();
atan();     mod();

sinh();     fabs();
cosh();     floor();
tanh();     ceil();
```

Συναρτήσεις κινητής υποδιαστολής
f_name(double x);

```
sin();      sqrtf();
cos();      powf();
tanf();
            expf();
asinf();    logf();
acosf();    log10f();
atanf();

sinhf();    fabs();
coshf();    floorf();
tanhf();    ceilf();
```

Συναρτήσεις διπλού τύπου
f_name(float x, floaty);

```
pow();
fmod();
atan2();
fmod();
```

Συναρτήσεις κινητής υποδιαστολής
f_name(double x, double y);

```
powf();
atan2f();
fmodf();
```