5. Εντολή while()

Η εντολή **while()** είναι ίσως η πιο πολυχρησιμοποιούμενη εντολή κατά τη σύνταξη κώδικα σε γλώσσα προγραμματισμού C για μικροελεγκτές. Το κυρίως μέρος του προγράμματος κλείνεται σχεδόν πάντα σε μια εντολή **while(1)** ώστε να εκτελείται συνεχώς ως επαναλαμβανόμενος βρόχος (loop). Η εντολή **while()** εκτελείται όταν η συνθήκη στην παρένθεση είναι αληθής (όταν είναι **1** είναι αληθής, **0** ψευδής). Η συνθήκη μπορεί να είναι ένας όρος, μια απλή ισότητα ή μια σύνθετη λογική πράξη. Όταν στο τέλος της εντολής υπάρχει ελληνικό ερωτηματικό (;) (semicolon), το πρόγραμμα εκτελεί την εντολή για όσο ισχύει η συνθήκη της παρένθεσης, πρακτικά ο επεξεργαστής δεν κάνει τίποτα και περιμένει να καταστεί η συνθήκη ψευδής ώστε να συνεχίσει την κανονική ροή του προγράμματος. Όταν δεν υπάρχει ελληνικό ερωτηματικό, αλλά κάτω από την εντολή υπάρχει μια άλλη εντολή, εκτελείτε αυτή για όσο ισχύει η συνθήκη της παρένθεσης, ενώ αν υπάρχουν αγκύλες εκτελείται το περιεχόμενο αυτών. Συνεπώς, όταν το κυρίως μέρος του προγράμματος κλείνεται σε αγκύλες κάτω από μια εντολή **while(1)**, τότε εκτελείται συνεχώς, αφού η συνθήκη είναι αληθής, είναι δηλαδή **1** και είναι σταθερή. Στη συνέχεια παρουσιάζονται παραδείγματα σύνταξης της εντολής:

```
while (PINA == 0b11101110);
```

Όταν το πρόγραμμα φτάσει σε αυτή την εντολή και ο καταχωρητής **PINA** έχει την τιμή **0b11101110** τότε την εκτελεί παραμένοντας στάσιμο σε αυτήν έως ότου αλλάξει η τιμή του καταχωρητή. Αν ο **PINA** έχει διαφορετική τιμή προσπερνά την εντολή χωρίς να την εκτελέσει. Σε αυτή την περίπτωση η συνθήκη στην παρένθεση είναι μια απλή ισότητα.

```
while ((key1 == 0xA0) && (key2 == 0xB0) && (key3 == 0xC0))
reg1 = 0;
```

Όταν το πρόγραμμα φτάσει σε αυτή την εντολή και οι μεταβλητές **key1, key2, key3** έχουν τις τιμές **0xA0**, **0xB0**, **0xC0** αντίστοιχα, τότε εκτελεί την επόμενη εντολή, η οποία είναι να αποδώσει την τιμή **0** στη μεταβλητή **reg1**. Αν κάποια από τις τρεις μεταβλητές έχει διαφορετική τιμή και οι δύο εντολές θα προσπεραστούν. Σε αυτή την περίπτωση η συνθήκη στην παρένθεση είναι λογικός συνδυασμός τριών ισοτήτων.

```
while (!(ADCSRA & (1 << ADIF)));
```

Όταν το πρόγραμμα φτάσει σε αυτή την εντολή και ο **ADIF** στον **ADCSRA** είναι **0** τότε την εκτελεί παραμένοντας στάσιμο σε αυτήν έως ότου αλλάξει η τιμή του **ADIF**. Σε αυτή την περίπτωση η συνθήκη μέσα στην παρένθεση είναι πιο σύνθετη. Ο **ADIF** είναι μια σημαία του καταχωρητή **ADCSRA**. Η παράσταση **(1 << ADIF)** προσδίδει λογικό **1** στον **ADIF** και η λογική πράξη **&** (AND) με τον **ADCSRA** θα επιστρέψει **1** αν ο **ADIF** του **ADCSRA** είναι και αυτός **1** ή θα επιστρέψει **0** αν ο **ADIF** του **ADCSRA** είναι **0**. Το θαυμαστικό μπροστά είναι το σύμβολο του αντίθετου. Η παραπάνω παράσταση δεν αλλάζει την πραγματική τιμή του **ADIF** στον **ADCSRA**, απλά επιστρέφει το αποτέλεσμα της πράξης. Τελικά η συνθήκη καθίσταται αληθής **(1)** όσο η πραγματική τιμή του **ADIF** στον **ADCSRA** είναι **0**.

Η ροή του προγράμματος μπορεί να βγει από μια **while()** χωρίς να καταστεί αληθής η συνθήκη της παρένθεσης με μια εντολή **break**:

```
i5 = 0;
while (PINA == 0b11101110)
{
    /* Your code here */
    i5++;
    _delay_us(5);
    if (i5 > 100)
    break;
}
```

Πριν την εντολή **while** υπάρχει μια εντολή όπου μηδενίζει την τιμή της μεταβλητής **i5**. Εφόσον ισχύει η συνθήκη της **while**, το πρόγραμμα εισέρχεται σε αυτή, εκτελεί όποιες εντολές υπάρχουν στο μπλοκ και αρχίζει να αυξάνει την **i5**. Η ρουτίνα στο μπλοκ εκτελείται συνεχώς όσο ισχύει η συνθήκη αυξάνοντας συνεχώς την τιμή της **i5** και όταν αυτή ξεπεράσει την τιμή **100** τότε εκτελεί την **break** η οποία βγάζει το πρόγραμμα από την **while** χωρίς να έχει μεταβληθεί ο **PINA**. Η τεχνική αυτή χρησιμοποιείται για να μην κρεμάει το πρόγραμμα όταν εισέρθει στην **while** και για οποιονδήποτε λόγο δεν μπορεί να μεταβληθεί συνθήκη της παρένθεσης. Η μέγιστη τιμή της μεταβλητής **i5** και η καθυστέρηση (***_delay_***) εξαρτώνται από το χρόνο που θέλουμε να επιτρέψουμε στο πρόγραμμα να μείνει μέσα στην **while**.

6. Εντολή if()

Η εντολή **if()** εκτελείται όταν η συνθήκη στην παρένθεση είναι αληθής (όταν είναι **1** είναι αληθής, **0** ψευδής). Η συνθήκη μπορεί να είναι ένας όρος, μια απλή ισότητα ή μια σύνθετη λογική πράξη. Όταν κάτω από την εντολή υπάρχει μια άλλη εντολή, εκτελείτε αυτή αν ισχύει η συνθήκη της παρένθεσης, ενώ αν υπάρχουν αγκύλες εκτελείται το περιεχόμενο αυτών. Η **if()** δε δημιουργεί επαναλαμβανόμενο βρόχο όπως δημιουργεί η **while**, αλλά οι εντολές που ακολουθούν εκτελούνται μια φορά και συνεχίζεται η κανονική ροή του προγράμματος. Παρακάτω παρουσιάζονται παραδείγματα σύνταξης της εντολής:

```
/* Your code here */
if (bit_is_set(PINB,0))           // if bit 0 of PORTB is 1 execute commands
{
    PORTA &= (1<<0);
    _delay_ms (1000);
}
/* Your code here */
```

Όταν το πρόγραμμα φτάσει στην εντολή **if(bit_is_set(PINB,0))** και το **bit 0** του **PORTB** είναι **1** τότε εκτελούνται οι εντολές στις αγκύλες. Στη συνέχεια εκτελούνται οι υπόλοιπες εντολές του κώδικα. Αν το **bit 0** του **PORTB** είναι **0** τότε αγνοούνται οι εντολές στις αγκύλες και το πρόγραμμα συνεχίζει με τις υπόλοιπες εντολές του κώδικα.

Η εντολή **if()** μπορεί να συνδυαστεί με τις **else if** και **else** σε περίπτωση που έχουμε μια πιο σύνθετη παραμετροποίηση με διαφορετικά σενάρια για διαφορετικές συνθήκες. Μια **if()** δεν είναι υποχρεωτικό να περιέχει στη δήλωσή της **else if** ή **else.** Αν υπάρχουν **else if** θα πρέπει να βρίσκονται πριν την **else**, αν αυτή υπάρχει, η οποία θα πρέπει να βρίσκεται πάντα στο τέλος. Αν κάποια συνθήκη επαληθευτεί δεν ελέγχονται οι υπόλοιπες.

```
/* Your code here */
if ((i1[0] == 3) && (i1[1] == 2) && (i1[2] == 6))      // if i1[]={3,2,6}
PORTA = 0b11111111;
else if ((i1[0] == 2) && (i1[1] == 1) && (i1[2] == 3)) // else if i1[]={2,1,3}
PORTA = 0b11110000;
else if ((i1[0] == 1) && (i1[1] == 0) && (i1[2] == 4)) // else if i1[]={1,0,4}
PORTA = 0b00001111;
else PORTA = 0b00000000;
/* Your code here */
```

Όταν το πρόγραμμα φτάσει στην εντολή **`if(…)`** και η μεταβλητή **i1**, η οποία είναι μονοδιάστατος πίνακας, έχει τις τιμές **`{3,2,6}`** το **PORTA** παίρνει τιμή **`0b11111111`**, αν έχει τις τιμές **`{2,1,3}`** το **PORTA** παίρνει τιμή **`0b11110000`**, αν έχει τις τιμές **`{1,0,4}`** το **PORTA** παίρνει τιμή **`0b00001111`** ενώ για οποιεσδήποτε άλλες τιμές της **i1**, παίρνει την τιμή **`0b00000000`**.

7. Εντολή for()

Η εντολή **for()** χρησιμοποιείται για να δημιουργήσουμε επαναλαμβανόμενους βρόχους (loops). Ο βρόχος μπορεί να είναι μια μόνο εντολή ή μια ομάδα εντολών κλεισμένες σε αγκύλες, αναλόγως τι ακολουθεί την εντολή. Η δήλωση μέσα στην παρένθεση περιλαμβάνει τρεις όρους: ο πρώτος καθορίζει την αρχική τιμή μιας μεταβλητής, ο δεύτερος καθορίζει τη συνθήκη σύγκρισης και ο τρίτος τη μεταβολή της μεταβλητής (αύξηση ή ελάττωση). Οι όροι χωρίζονται μεταξύ τους με το σύμβολο **;** (ελληνικό ερωτηματικό). Η μεταβλητή πρέπει να δηλωθεί είτε μέσα στη δήλωση της **for()**, είτε εξωτερικά. Ο βρόχος επαναλαμβάνεται έως ότου ικανοποιηθεί η συνθήκη, εκτός και αν διακοπεί από μια **break**, αν αυτή υπάρχει με κάποια συνθήκη στο μπλοκ της **for**. Στη συνέχεια παρουσιάζονται παραδείγματα σύνταξης της εντολής:

```
/* Your code here */
for (uint8_t i1=0; i1<7; i1++)           // execute loop 7 times
{
    i2 = i2<<1;                          // i2 shift left
    PORTA = ~i2;                         // PORTA complement i2
    _delay_ms(100);                      // delay 100ms
}
/* Your code here */
```

Όταν το πρόγραμμα φτάσει στην εντολή **for(…)** δημιουργεί τη μεταβλητή **i1** με αρχική τιμή **0**, ελέγχει τη μεταβλητή να ικανοποιεί τη συνθήκη **i1<7**, εκτελεί τις εντολές μέσα στις αγκύλες και αυξάνει τον **i1** κατά 1. Στον επόμενο κύκλο ελέγχει ξανά τη μεταβλητή να ικανοποιεί τη συνθήκη **i1<7**, εκτελεί ξανά τις εντολές, αυξάνει τον **i1** κατά 1 και ο κύκλος επαναλαμβάνεται για όσο ικανοποιείται η συνθήκη, στο συγκεκριμένο παράδειγμα 7 φορές.

8. Εντολή `switch()`

Η εντολή **`switch()`** χρησιμοποιείται για να επιλέξουμε από ένα σύνολο τιμών αυτές που ικανοποιούν τα σενάριά μας για μία μεταβλητή. Η μεταβλητή γράφεται στην παρένθεση της **`switch`** και η τιμή που επιλέγει κάθε σενάριο είναι η **`case`** και πρέπει να είναι του ίδιου τύπου με τη μεταβλητή. Η μεταβλητή πρέπει να έχει αριθμητικό χαρακτήρα. Όταν επιλεγεί μια **`case`** εκτελούνται οι εντολές κάτω από αυτήν και στη συνέχεια το πρόγραμμα εξέρχεται από τη **`switch`** με μια **`break`**. Η **`break`** δεν είναι απαραίτητο να υπάρχει σε κάθε **`case`**. Όταν μια **`case`** δεν έχει **`break`** και επιλεγεί, εκτελούνται οι εντολές της και στη συνέχεια οι εντολές των επόμενων **`case`**, μέχρι να προκύψει μια **`break`**. Στο τέλος μπορεί να υπάρχει μια **`default`** ώστε να συμπεριλάβει όλες τις υπόλοιπες περιπτώσεις αν κάτι τέτοιο είναι απαραίτητο. Η **`default`** δεν είναι απαραίτητη και δε χρειάζεται **`break`**. Αν επιλεγεί μια **`case`** δεν εξετάζεται η **`default`**. Στη συνέχεια παρουσιάζονται παραδείγματα σύνταξης της εντολής:

```
/* Your code here */
switch (PINA)
{
    case 0b11111110:              // Button 1 pressed
    i1 = 0;
    data = 1;
    while (PINA == 0b11111110);   // Wait if button is still pressed
    break;

    case 0b11111101:              // Button 2 pressed
    i1 = 1;

    case 0b11111011:              // Button 3 pressed
    i1 = 2;
    data = 0xFF;
    while (PINA == 0b11111011);   // Wait if button is still pressed
    break;

    default:
    data = 0;
}
/* Your code here */
```

Όταν το πρόγραμμα φτάσει στην **switch** ελέγχει τις **case**. Αν το περιεχόμενο του **PINA** είναι **0b11111110** ικανοποιείται η πρώτη **case**, η οποία και επιλέγεται και εκτελούνται οι εντολές κάτω από αυτήν. Στη συνέχεια το πρόγραμμα εξέρχεται από τη **switch** λόγω της **break**. Αν το περιεχόμενο του **PINA** είναι **0b11111101** ικανοποιείται η δεύτερη **case** και εκτελείται η εντολή κάτω από αυτήν και στη συνέχεια οι επόμενες εντολές, οι οποίες βρίσκονται κάτω από την τρίτη **case** και το πρόγραμμα εξέρχεται από τη **switch** λόγω της **break** σε αυτήν. Αν το περιεχόμενο του **PINA** είναι **0b11111011** ικανοποιείται η τρίτη **case** και εκτελούνται οι εντολές κάτω από αυτήν. Στη συνέχεια το πρόγραμμα εξέρχεται από τη **switch** λόγω της **break**. Αν δεν επιλεγεί καμία **case** εκτελούνται οι εντολές κάτω από την **default** και στη συνέχεια το πρόγραμμα εξέρχεται από την **switch**. Αν δεν υπάρχει **default** το πρόγραμμα εξέρχεται από την **switch** χωρίς να εκτελέσει καμία εντολή.

9. Εντολή `return`

Η εντολή **`return`** επιστρέφει στη ρουτίνα που καλείται μια τιμή η οποία μπορεί να είναι μια σταθερά ή η τιμή μιας μεταβλητής. Δεν απαιτείται να υπάρχει μέσα στις ρουτίνες ή στο κυρίως πρόγραμμα γιατί η λειτουργία της μπορεί να αντικατασταθεί με απόδοση τιμής σε μια μεταβλητή. Στη συνέχεια παρουσιάζεται ένα παράδειγμα σύνταξης με την εντολή **`return`**:

```
uint8_t   j, k, m;

int min(uint8_t a, uint8_t b)
{
    if (a < b)
    return a;
    else
    return b;
}

int main(void)
{
    while (1)
    {
              /* Your code here */
        m = min(j, k);
              /* Your code here */
    }
}
```

Το παραπάνω πρόγραμμα συγκρίνει τις μεταβλητές **`j, k`** μεταξύ τους και αποδίδει τη μικρότερη τιμή στη μεταβλητή **`m`**. Η απόδοση γίνεται στο κυρίως πρόγραμμα με την εντολή **`m=min(j, k);`** όπου γίνεται κλήση της ρουτίνας **`int min(…)`**, στην οποία η **`return`** επιστρέφει το αποτέλεσμα της σύγκρισης. Το συγκεκριμένο πρόγραμμα μπορεί να γραφεί με διαφορετικό τρόπο ως εξής:

```
uint8_t   j, k, m;

int min(uint8_t a, uint8_t b)
{
    if (a < b)
```

```
    m = a;
    else
    m = b;
}

int main(void)
{
    while (1)
    {
              /* Your code here */
        min(j, k);
              /* Your code here */
    }
}
```

Όταν στο κυρίως πρόγραμμα γίνει κλήση της ρουτίνας **`int min(…)`** θα αποδοθεί στη μεταβλητή **`m`** η μικρότερη τιμή από τις **`j, k`**. Τα δύο προγράμματα είναι ισοδύναμα.