

Αναφορά SUDOKU

Εισαγωγή :

Σκοπός της αυτού του project είναι η δημιουργία της εφαρμογής που θα τρέχει πάνω στο μικροελεγκτή ATmega16 της πλακέτας STK500 και θα δίνει την λύση για κάθε SUDOKU που θα λαμβάνει στην είσοδο του με την κατάλληλη διεπαφή όσο το δυνατό γρηγορότερα. Πέρα από τον αλγόριθμο μετάδοσης δεδομένων μέσω της σειριακής θύρας που έχει δημιουργηθεί, έχει υλοποιηθεί και ο αλγόριθμος επίλυσης του Sudoku με μέθοδο backtrack. Ακόμα, υπάρχει κώδικας που είναι υπεύθυνος για την ενεργοποίηση των LEDS ανάλογα με την πρόοδο της επίλυσης του Sudoku με τη χρήση ενός Timer/Counter και του interrupt που αυτός εκτελεί κάθε διάστημα 40 ms .Όλος ο κώδικας γράφτηκε σε γλώσσα C στο περιβάλλον Atmel studio για μικροελεγκτή ATmega16.

Εκτέλεση:

Υλοποίηση σειριακής διεπαφής :

1. Για την υλοποίηση του προγράμματος αρχικά ορίστηκε σαν global ο 9x9 πίνακας grid που θα κρατά τις τιμές του κάθε κελιού του Sudoku , η μεταβλητή progress που θα κρατά τον αριθμό των κελιών που είναι λυμένα ,το flag play που θα ανακοινώνει την έναρξη της επίλυσης του , ο πίνακας readWord που θα κρατά κάθε χαρακτήρα που λαμβάνεται από τη σειριακή μέχρι το <LF> και κάποιες μεταβλητές που παίζουν το ρόλο counter για να βρίσκουμε τη κατάλληλη θέση επεξεργασίας των παραπάνω πινάκων .

2. Με την έναρξη του προγράμματος πρώτα εκτελείται η συνάρτηση `main` όπου και αρχικοποιούνται στο 0 οι `global` μεταβλητές που αναφέρθηκαν παραπάνω . Μετά την αρχικοποίηση τίθεται το `PortB` ως `output` όπου θα φαίνεται η πρόοδος της επίλυσης Sudoku μέσω των `led` ,ενεργοποιείται το `flag` των `interrupt` με την εντολή `SEI()` , τίθεται σε λειτουργία ο `TIMER2` για την ανανέωση των `led` με βάση το `progress` κάθε 40 ms μέσω της εκτέλεσης του `ISR` του, καλείται η υπορουτίνα `initUART()` για την αρχικοποίηση της σειριακής θύρας και τέλος υλοποιείται ένας ατέρμονος βρόχος ο οποίος ελέγχει το `flag play` και σε περίπτωση που είναι ενεργό λύνει το Sudoku .

3. Η ρουτίνα `initUART()` που καλείται στη συνάρτηση `main` λίγο πριν την έναρξη του ατέρμονου βρόχου αρχικοποιεί τη σειριακή θύρα ενεργοποιώντας στο `register UCSRA` τα `flags RXEN` και `TXEN` για να μπορεί να γίνει ανάγνωση και αποστολή δεδομένων καθώς και το `RXCIE` για να ενεργοποιηθούν τα `interrupt` του `UART` και το πρόγραμμα να μεταβαίνει στο κατάλληλο `ISR` όπου θα γίνονται οι απαραίτητες ενέργειες . Επίσης τίθεται το `UBRR1` στην κατάλληλη τιμή για την επίτευξη του 9600 baud rate που για 10 Mhz είναι η τιμή 64 η οποία προκύπτει από τον τύπο $UBRR1 = 10^7 / 9600 / 16 - 1$ και τέλος ενεργοποιούνται τα `flags UCSZ0` και `UCSZ1` του καταχωρητή `UCSRC` για τον ορισμό του αριθμού των bit που μεταδίδονται στα 8.

4. Στη συνέχεια υλοποιήθηκε το `ISR` του `UART` το οποίο εκτελείται κάθε φορά που το πρόγραμμα λαμβάνει ένα χαρακτήρα από την σειριακή . Αρχικά μέσω ενός βρόχου που εκτελείται όσο το `RXC` είναι 1 δηλαδή όσο υπάρχει νέος χαρακτήρας από τη σειριακή γίνεται η ανάγνωση μέσω της υπορουτίνας `getC` η οποία παίρνει το χαρακτήρα από το `UDR` και τον τοποθετεί στην επόμενη ελεύθερη θέση του `global` πίνακα `readWord` . Στη συνέχεια αφού το πρόγραμμα βγει από το βρόχο ελέγχεται αν ο τελευταίος χαρακτήρας που διαβάστηκε και έχει επιστραφεί από την `getC` είναι ο `<LF> = 0x0A` . Αν η συνθήκη είναι ψευδής το τρέχον `interrupt` τερματίζει διαφορετικά ακολουθεί μια διαδικασία συνεχόμενων ελέγχων για το ποια ενέργεια θα εκτελεστεί σύμφωνα με το προβλεπόμενο πρωτόκολλο . Αν η λέξη `readWord` που διαβάστηκε είναι η `AT` τότε καλείται η υπορουτίνα `putC` 4 φορές , η οποία στέλνει ένα χαρακτήρα τη φορά μέσω σειριακής ,και αποστέλλεται η λέξη `OK<CR><LF>` . Στην περίπτωση που ληφθεί η λέξη `C<CR><LF>` από τη σειριακή μηδενίζονται όλες οι `global` μεταβλητές που έχουν οριστεί και αποστέλλεται με αντίστοιχο τρόπο η λέξη `OK<CR><LF>` . Σε περίπτωση που η λέξη που ληφθεί η λέξη `N<Xpos><Ypos><Value><CR><LF>` όπου `Xpos`,`Ypos`,`Value` αριθμοί μεταξύ 1-9 ,γίνεται η μετατροπή των αριθμών από `ASCII` σε δεκαδικούς , τοποθετείται το `Value`

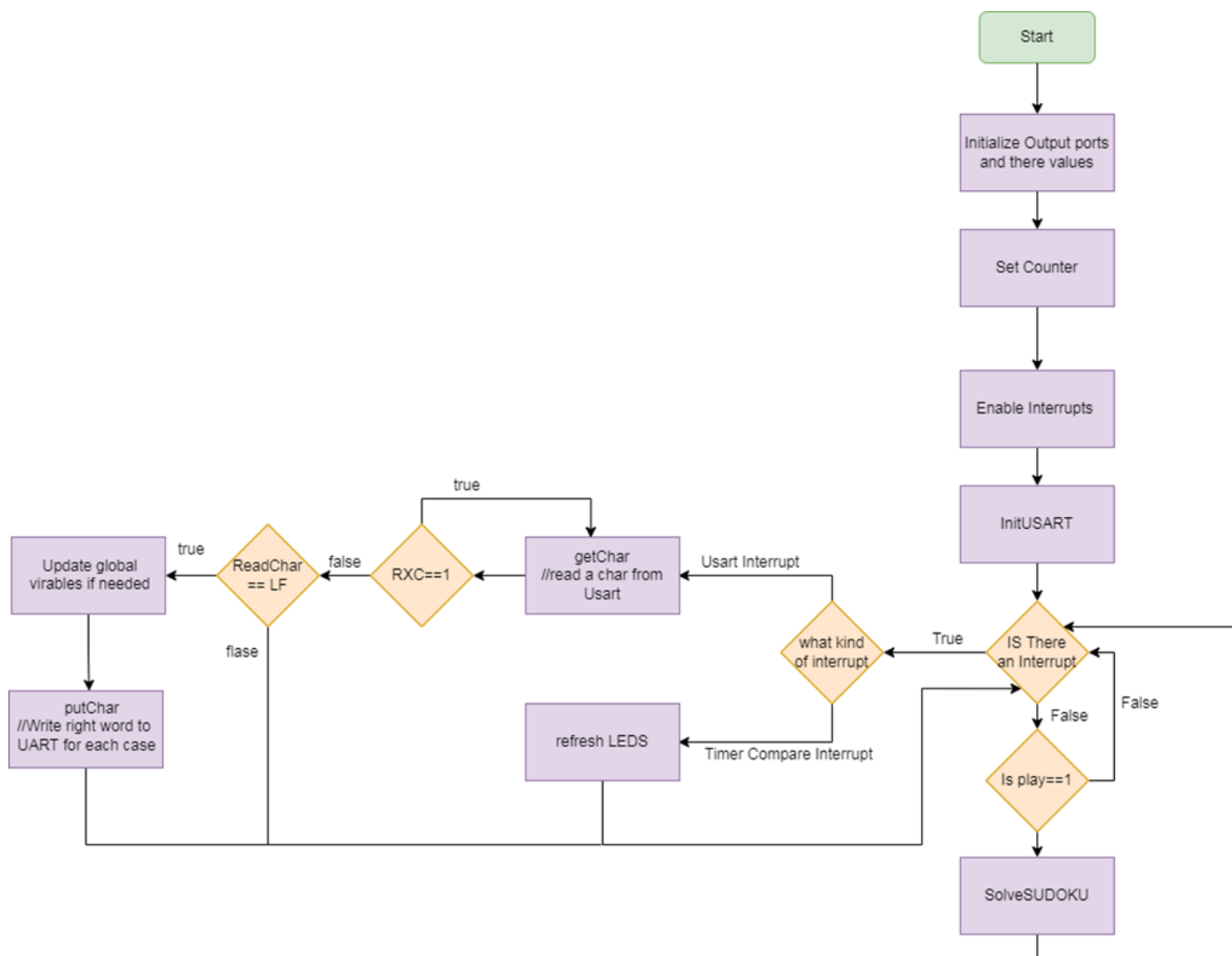
στις αντίστοιχες συντεταγμένες Χpos,Υpos στον πίνακα Grid και αποστέλλεται πίσω η λέξη OK<CR><LF>. Σε περίπτωση που ληφθεί η λέξη P<CR><LF> ενεργοποιείται το flag play ώστε να ξεκινήσει η επίλυση του Sudoku όταν το πρόγραμμα επιστρέψει στην συνάρτηση main και αποστέλλεται η λέξη D<CR><LF> όταν και αν ο αλγόριθμος βρει λύση για το Sudoku. Σε περίπτωση που ληφθεί η λέξη S<CR><LF> αποστέλλεται το πρώτο ψηφίο της λύσης του Sudoku μέσω της λέξης N11<Grid[0][0]><CR><LF>. Σε περίπτωση που ληφθεί η λέξη T<CR><LF> αποστέλλεται πίσω το επόμενο στοιχείο του πίνακα από αυτό που αποστάλθηκε την τελευταία φορά μέσω της λέξης N<Χpos><Υpos><Value><CR><LF> και σε περίπτωση που έχουν αποσταλεί όλα τα στοιχεία του πίνακα στέλνεται μόνο η λέξη D<CR><LF>. Σε περίπτωση που ληφθεί η λέξη B<CR><LF> γίνεται το flag play =0 έτσι ώστε το πρόγραμμα να σταματήσει την εκτέλεση του Sudoku εφόσον αυτή εκτελείται και αποστέλλεται πίσω πάλι με αντίστοιχο τρόπο η λέξη OK<CR><LF> .. Σε περίπτωση που η λέξη που ληφθεί η λέξη D<Χpos><Υpos><CR><LF> αποστέλλεται πίσω η λέξη N<Χpos><Υpos><Value><CR><LF> όπου Χpos Υpos οι συντεταγμένες του πίνακα που ζήτησε ο χρήστης και Value η τιμή για τις αντίστοιχες συντεταγμένες . Αφού σταλεί μία από τις παραπάνω λέξεις μηδενίζεται ο πίνακας readWord και ο απαραίτητος counter για την τοποθέτηση των ψηφίων στην κατάλληλη θέση ώστε να είναι έτοιμη η κατάσταση λήψης της επόμενης .

Σειμώση : ο αλγόριθμος στέλνει όλους τους χαρακτήρες απάντησης από τη σειριακή θύρα (πχ OK<CR><LF>) χωρίς να βγει απο το interrupt καθώς δεν υπάρχει κάτι χρήσιμο να κάνει το πρόγραμμα στο ενδιάμεσο αφού είναι μέσα σε ένα ατέρμονο βρόγχο στη συνάρτηση main και το πρωτόκολλο που χρησιμοποιείται υποστηρίζει αποστολή της επόμενης λέξης μόνο αφού έχει ληφθεί απάντηση για την απεσταλμένη λέξη. Εξαιρέση αποτελεί η απάντηση στη λέξη P<CR><LF> που ξεκινά την επίλυση του Sudoku καθώς το πρωτόκολλο προβλέπει αποστολή της λέξης B<CR><LF> και D<CR><LF> προτού ληφθεί απάντηση κάτι που δεν θα μπορούσε να γίνει σωστά αν το πρόγραμμα παρέμενε μέσα στο interrupt .

Υλοποίηση Timer για ανανέωση των LEDs :

Όπως προαναφέρθηκε παραπάνω στην αρχή της συνάρτησης main τίθεται το PORTB ως output το οποίο είναι συνδεδεμένο με τα led της πλακέτας. Ακόμα ενεργοποιείται το bit

OCIE1A του register TIMSK για την ενεργοποίηση των compare interrupts του Timer, ενεργοποιείται το CS12 για τον ορισμό του prescaler στο 256 και τίθεται η τιμή OCR1A =1536 ώστε να εκτελείται ένα interrupt κάθε $1536 \cdot 256 / 10000000 = 0,0393216 = 40\text{ms}$. Όταν λοιπόν εκτελείται το interrupt timer εισέρχονται οι κατάλληλοι καταχωρητές στη στοίβα και με βάση τη μεταβλητή progress, που έχει αρχικοποιηθεί ως global και αυξάνεται από τον αλγόριθμο του Sudoku όποτε επιλύεται ένα καινούριο κελί, σβήνει το αντίστοιχο led για κάθε δεκάδα αύξησης του progress. Τέλος γίνονται pop οι registers που είχαν μπει στη στοίβα και το πρόγραμμα συνεχίζει την εκτέλεση του από εκεί που σταμάτησε.



Υλοποίηση backtrack αλγόριθμος για Sudoku:

Ο αλγόριθμος επίλυσης του sudoku βασίζεται στο backtracking και γράφτηκε και αυτός σε κώδικα C. Η επίλυση γίνεται μέσω της αναδρομικής συνάρτησης SolveSudoku(). Με την κλήση της συνάρτησης γίνεται έλεγχος στο grid του sudoku με σκοπό την εύρεση θέσης στην οποία υπάρχει η τιμή 0 η οποία έχει ορισθεί ως τιμή για unassigned θέσεις (κλήση της συνάρτησης FindUnassignedLocation()). Σε περίπτωση εύρεσης θέσης που βρίσκεται στην κατάσταση unassigned, μέσω επαναληπτικής διαδικασίας ο αλγόριθμος τοποθετεί σειριακά αριθμούς από το ένα μέχρι το εννιά και ελέγχει αν η τοποθέτηση του συγκεκριμένου αριθμού στην παρούσα θέση δεν παραβιάζει τις συνθήκες παιχνιδιού του sudoku. Σε περίπτωση επιτυχίας, δηλαδή σε περίπτωση που βρεθεί ο αριθμός ο οποίος έχει την δυνατότητα να βρίσκεται στην θέση την οποία εξετάζει η συνάρτηση (κάτι που υλοποιείται μέσω της κλήσης της συνάρτησης isSafe()), τότε ανατίθεται σε αυτήν και η συνάρτηση καλείται αναδρομικά για την επόμενη unassigned θέση μέχρι το grid να γεμίσει, που συνεπάγεται σε λύση του sudoku. Σε κάθε άλλη περίπτωση, η θέση ξανα θεωρείται unassigned και γίνεται δοκιμή άλλου αριθμού. Σε περίπτωση που δεν βρεθεί αριθμός που να ταιριάζει στην συγκεκριμένη θέση η συνάρτηση επιστρέφει 0 και το sudoku κρίνεται μη επιλύσιμο οπότε ο αλγόριθμος γυρίζει σε προηγούμενο αριθμό. Κάθε αριθμός ο οποίος προστίθεται επιτυχώς στο grid αυξάνει το progress επίλυσης του κατά ένα, κάτι που είναι σημαντικό για την ένδειξη των LED. Συνθήκη εξόδου της solveSudoku() είναι η περίπτωση που δεν υπάρχουν άλλες unassigned θέσεις, κάτι που συνεπάγεται σε λύση του sudoku, είτε να έχει έρθει από τον χρήστη η εντολή break η οποία όπως έχει περιγραφεί κάνει το play flag 0.

Συνάρτηση FindUnassignedLocation()

Η συνάρτηση FindUnassignedLocation() έχει την εξής λειτουργία:

Ξεκινώντας από την αρχική θέση του grid (θέση 0,0) μέσω μίας επαναληπτικής διαδικασίας γίνεται έλεγχος για την πρώτη θέση που θα βρεθεί στο grid με την τιμή unassigned, δηλαδή 0. Πιο αναλυτικά, μέσω μίας επαναληπτικής διαδικασίας while, η οποία τρέχει στο μήκος των γραμμών έχει τοποθετηθεί μία εμφωλευμένη while η οποία τρέχει στο μήκος των στηλών. Με αυτόν τον τρόπο γίνεται έλεγχος όλων των στοιχείων κάθε γραμμής μέχρι να βρεθεί το επιθυμητό αποτέλεσμα (θέση με unassigned τιμή). Αν βρεθεί τέτοια θέση, η συνάρτηση επιστρέφει 1(true) και επειδή τόσο η μεταβλητή υπεύθυνη για την γραμμή που βρίσκεται η συνάρτηση όσο και η στήλη τρέχουν μέσω

pointer η θέση γίνεται update επίσης. Σε περίπτωση που δεν βρεθεί θέση που να έχει unassigned τιμή, τότε η συνάρτηση επιστρέφει μηδέν.

Συνάρτηση isSafe()


Η συνάρτηση isSafe υπάρχει με σκοπό της διασφάλισης ότι ο αριθμός που πρόκειται να μπει σε μία συγκεκριμένη θέση του grid τηρεί τις προϋποθέσεις του παιχνιδιού. Έχει ως ορίσματα την θέση του grid σε γραμμή, στήλη στην οποία θα γίνει ο έλεγχος και τον αριθμό ο οποίος θα ελεγχθεί. Οι προϋποθέσεις είναι οι εξής:

- Δεν υπάρχει ο ίδιος αριθμός κατα μήκος της γραμμής που βρίσκεται η θέση στην οποία θα τοποθετηθεί.
- Δεν υπάρχει ο ίδιος αριθμός κατα μήκος της στήλης που βρίσκεται η θέση στην οποία θα τοποθετηθεί.
- Δεν υπάρχει ο ίδιος αριθμός στο sub-grid της θέσης στην οποία θα τοποθετηθεί. Sub-grid θεωρείται το grid το οποίο είναι διαστάσεων 3x3.

Για να γίνεται ο έλεγχος που περιγράφεται παραπάνω, τίθεται μία μεταβλητή safe η οποία θα χρησιμοποιηθεί σαν flag. Σε περίπτωση που το safe είναι μηδέν σημαίνει ότι κάποια συνθήκη από τις παραπάνω παραβιάζεται.

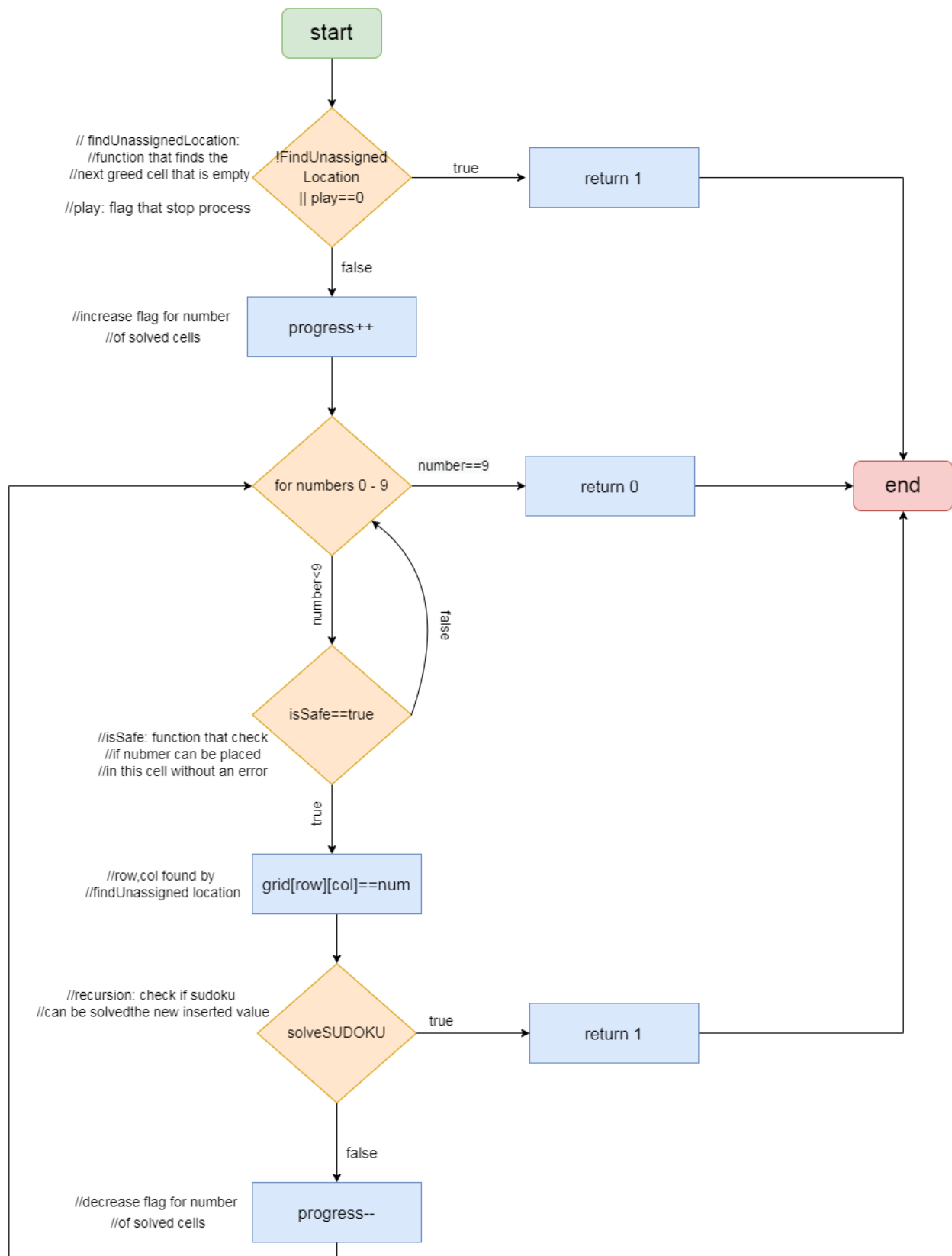
Για να μην υπάρχει αλλαγή στις τιμές των ορισμάτων θέσης χρησιμοποιήθηκαν οι μεταβλητές tempRow, tempCol. Τόσο ο πρώτος όσο και ο δεύτερος έλεγχος έγιναν με τον ίδιο τρόπο. Μέσω μίας επαναληπτικής διαδικασίας while ελέγχεται κάθε φορά αν υπάρχει κάποια θέση στην ίδια γραμμή/στήλη η οποία έχει τον ίδιο αριθμό. Επιπλέον ελέγχεται πάντα αν το safe flag έχει γίνει 1. Σε περίπτωση που βρεθεί ίδιος αριθμός στην ίδια γραμμή/στήλη τότε η μεταβλητή safe αλλάζει σε 0 και η αντίστοιχη επαναληπτική διαδικασία σταματά μη επιτρέποντας και στις επόμενες να ξεκινήσουν.

Τέλος, για τον έλεγχο του αν υπάρχει ο ίδιος αριθμός σε ένα sub-grid. Για να βρεθεί το αντίστοιχο sub-grid η θέση του αριθμού (και οι στήλη και οι γραμμή) γίνονται modulo 3 και τοποθετούνται στις μεταβλητές tempCol και tempRow αντίστοιχα. Έπειτα μέσω μίας διπλής επαναληπτικής διαδικασίας που τρέχει από το 0 μέχρι το 2 τόσο στις γραμμές όσο και στις στήλες γίνεται έλεγχος για την περίπτωση που υπάρχει κάποια θέση μέσα σε

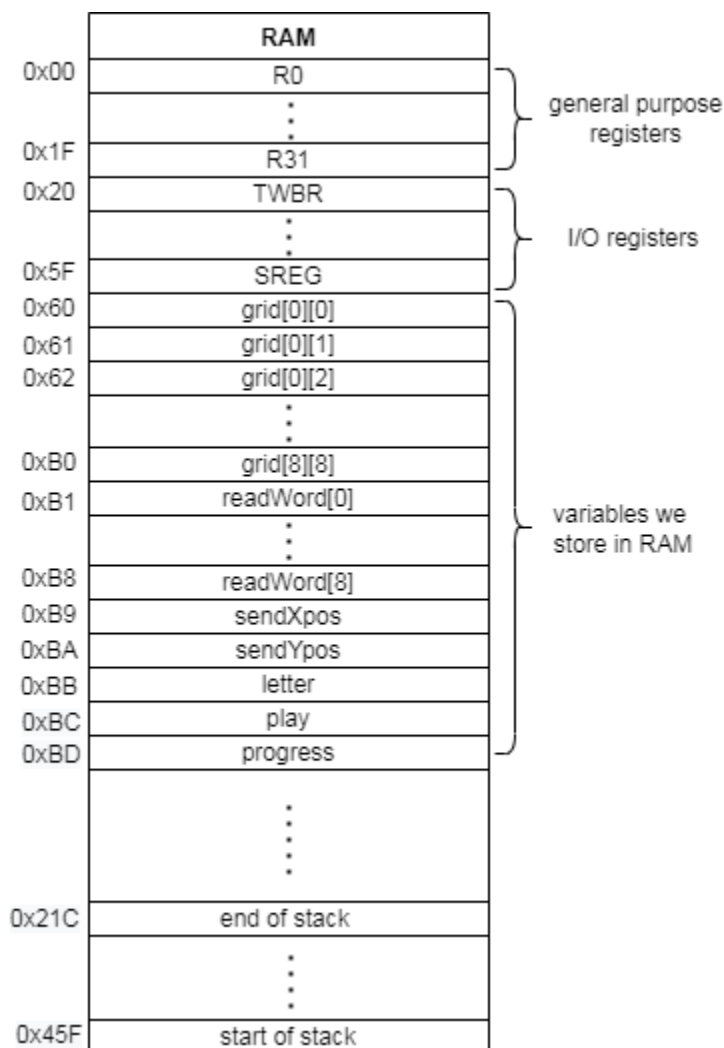


αυτές η οποία να έχει τιμή ίση με αυτή του αριθμού που υπάρχει ως όρισμα. Αν υπάρχει τέτοια περίπτωση, τότε η μεταβλητή `safe` γίνεται 0. Σε κάθε άλλη περίπτωση θεωρώντας ότι η συνάρτηση έχει εκτελέσει ήδη τις δύο προηγούμενες `while` και το `safe flag` έχει παραμείνει ένα τότε ο αλγόριθμος επιστρέφει ένα, σημαίνοντας ότι υπάρχει δυνατότητα να τοποθετηθεί ο αριθμός στην συγκεκριμένη θέση.

solveSUDOKU



Αποτελέσματα:



Όπως φαίνεται και στο διπλανό σχήμα η μνήμη έχει στις θέσεις από 0x00 - 0x1F τους καταχωρητές γενικής χρήσης τους οποίους χρησιμοποιεί ο compiler με όποιο τρόπο θέλει για την αποθήκευση μίας τιμής προσωρινά. Στη συνέχεια από τη θέση 0x20 - 0x5F βρίσκονται οι καταχωρητές Input / Output όπως το PORTA και PORTC που χρησιμοποιήθηκαν και γενικότερα όλους τους default καταχωρητές με τα flags που ενεργοποιούν κάποια συγκεκριμένη λειτουργία. Έπειτα, από την θέση 0x60 και μετά αποθηκεύονται τα δεδομένα που το πρόγραμμα χρησιμοποιεί για μεγάλο χρονικό διάστημα η που έχουν οριστεί ως global τις οποίες ο compiler προτιμά να τα αποθηκεύσει εκεί από το να σπαταλάει καταχωρητές γενικής

χρήσης που είναι χρήσιμοι για περιστασιακές ενέργειες. Για το συγκεκριμένο πρόγραμμα ο compiler αποφασίζει να αποθηκεύσει τον global πίνακα grid που κρατά τις τιμές κάθε κελιού του sudoku στις διευθύνσεις από 0x60 - 0xB0. Ακόμα αποθηκεύονται ο πίνακας readWord που κρατά τους χαρακτήρες που λαμβάνονται από τη σειριακή θύρα καθώς και την μεταβλητή letter που δείχνει τη θέση του πίνακα που θα μπει ο επόμενος χαρακτήρας στις επόμενες θέσεις διαδοχικά μετά το τελευταίο στοιχείο του grid. Επίσης αποθηκεύονται διαδοχικά και οι μεταβλητές sendXpos, sendYpos για τις επόμενες τιμές του grid που θα σταλούν αφού έχει λυθεί ο αλγόριθμος καθώς και οι μεταβλητές play, progress που χρησιμοποιούνται για την εκκίνηση και πρόοδο της εκτέλεσης του Sudoku. Τέλος όπως φαίνεται στο σχήμα στην τελευταία διεύθυνση της RAM είναι η αρχή

του stack pointer ο οποίος κάθε φορά που γίνεται push η Pop μειώνεται ή αυξάνεται κατά 1 αντίστοιχα. Η μεγαλύτερη τιμή μνήμης που φτάνει ο stack pointer που στην χειρότερη περίπτωση δηλαδή όταν το παζλ λυθεί με κανένα στοιχείο ως αρχική τιμή και άρα γίνονται 81 αναδρομές είναι η 0x21C.

1st Test:

Στα παρακάτω τεστ φαίνεται δεξιά το παζλ εισόδου ,εξόδου που δίνεται και λαμβάνεται από το πρόγραμμα και δεξιά η επικοινωνία για την αρχικοποίηση και λύση μέσω putty.

```
COM4 - PuTTY
AT^M^J
OK
C^M^J
OK
N112^M^J
OK
N998^M^J
OK
N555^M^J
OK
P^M^J
OK
D
S^M^J
N112
T^M^J
N121
T^M^J
N133
T^M^J
N144
T^M^J
N156
T^M^J
N165
T^M^J
N177
T^M^J
N188
D78^M^J
N784
C^M^J
OK
S^M^J
N110
T^M^J
N120
```

2								
								8

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


2nd Test:

P^M^J
 OK
 D
 S^M^J
 N118
 T^M^J
 N121
 T^M^J
 N132
 T^M^J
 N147
 T^M^J
 N155
 T^M^J
 N163
 T^M^J
 N176
 T^M^J
 N184
 T^M^J
 N199
 T^M^J
 N219
 D78^M^J
 N786
 D99^M^J
 N992

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

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

Όπως φαίνεται στις παραπάνω φωτογραφίες, ο ελεγκτής ανταποκρίνεται κανονικά στα σήματα ελέγχου που του δίνονται. Όπως φαίνεται, σε όλες τις εντολές εισόδου ο ελεγκτής



απαντά με OK. Μετά την εντολή P ο ελεγκτής ανταποκρίνεται με D που συνεπάγεται σε λύση του sudoku. Επιπλέον φαίνεται ότι ο ελεγκτής ανταποκρίνεται κανονικά στις εντολές T, S, D.

Simulations με το πρόγραμμα putty:

Όπως φαίνεται παραπάνω ο ελεγκτής επιστρέφει τόσο στο πρώτο πείραμα όσο και στο δεύτερο σωστές λύσεις του sudoku. Για το χρονικό περιθώριο λόγω του ότι δεν υπήρχε δυνατότητα εκτέλεσης του προγράμματος ο χρόνος μας είναι λίγο ανακριβής. Για το πρώτο test ο χρόνος είναι ms και το δεύτερο test περίπου 9 sec.

Simulation με την χρήση του βοηθητικού προγράμματος:

Το πρόγραμμα έτρεξε σε λειτουργικό Ubuntu 20.04.3 LTS καθώς η έκδοση που μας δόθηκε για λειτουργικό σύστημα Windows δεν ήταν λειτουργική στα δικά μας μηχανήματα τουλάχιστον. Δοκιμάστηκαν οι εξής περιπτώσεις: Easy mode, Medium mode, Hard mode, Ultra mode. Όλα τα modes επιλύθηκαν επιτυχώς. Τα χρονικά αποτελέσματα για κάθε mode παρουσιάζονται παρακάτω με μορφή εικόνων:

Printing Unsolved Board:

Difficulty: EASY

Filled Cells: 46

```

-----
1 | _ 9 1 | 3 6 _ | 8 _ 7 |
2 | 5 8 7 | 1 2 _ | 4 _ 6 |
3 | _ 3 _ | _ _ 8 | 1 9 _ |
=====
4 | 9 _ 8 | 4 _ _ | 2 _ 3 |
5 | 3 2 4 | _ _ _ | _ _ _ |
6 | 7 6 5 | _ _ _ | 9 _ 4 |
=====
7 | _ 5 _ | 6 _ _ | 7 4 _ |
8 | _ _ _ | 5 7 1 | 3 6 _ |
9 | _ 7 3 | 8 9 4 | 5 _ 2 |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

18:01:23 [INFO] Generated Solution!

Printing Solved Board!

Difficulty: EASY

```

-----
1 | 4 9 1 | 3 6 5 | 8 2 7 |
2 | 5 8 7 | 1 2 9 | 4 3 6 |
3 | 2 3 6 | 7 4 8 | 1 9 5 |
=====
4 | 9 1 8 | 4 5 6 | 2 7 3 |
5 | 3 2 4 | 9 8 7 | 6 5 1 |
6 | 7 6 5 | 2 1 3 | 9 8 4 |
=====
7 | 1 5 9 | 6 3 2 | 7 4 8 |
8 | 8 4 2 | 5 7 1 | 3 6 9 |
9 | 6 7 3 | 8 9 4 | 5 1 2 |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

> play

18:02:08 [INFO] Writing 3 bytes to /dev/ttyUSB0

18:02:08 [INFO] Wrote 3 bytes!

18:02:08 [INFO] Bytes Read: [79, 75, 13, 10]

18:02:08 [INFO] Bytes Read: [68, 13, 10]

18:02:08 [INFO] Solved in: 7.825889ms

Printing Unsolved Board:

Difficulty: MEDIUM

Filled Cells: 41

```

-----
1 | 4 2 _ | 8 6 9 | _ 3 _ |
2 | _ 1 _ | _ 4 _ | 9 _ _ |
3 | 9 3 _ | _ _ _ | 8 _ _ |
=====
4 | 5 6 8 | _ _ _ | 2 9 _ |
5 | _ _ _ | _ 2 _ | _ 8 _ |
6 | _ _ 2 | _ _ 8 | 6 _ 1 |
=====
7 | _ 8 9 | 4 1 5 | _ _ 2 |
8 | 6 5 1 | 9 _ 2 | 4 _ 8 |
9 | _ 4 _ | _ 8 _ | 5 1 9 |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

18:08:37 [INFO] Generated Solution!

Printing Solved Board!

Difficulty: MEDIUM

```

-----
1 | 4 2 5 | 8 6 9 | 1 3 7 |
2 | 8 1 7 | 2 4 3 | 9 5 6 |
3 | 9 3 6 | 7 5 1 | 8 2 4 |
=====
4 | 5 6 8 | 1 7 4 | 2 9 3 |
5 | 1 9 4 | 3 2 6 | 7 8 5 |
6 | 3 7 2 | 5 9 8 | 6 4 1 |
=====
7 | 7 8 9 | 4 1 5 | 3 6 2 |
8 | 6 5 1 | 9 3 2 | 4 7 8 |
9 | 2 4 3 | 6 8 7 | 5 1 9 |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

> play

18:06:45 [INFO] Writing 3 bytes to /dev/ttyUSB0

18:06:45 [INFO] Wrote 3 bytes!

18:06:45 [INFO] Bytes Read: [79, 75, 13, 10]

18:06:45 [INFO] Bytes Read: [68, 13, 10]

18:06:45 [INFO] Solved in: 77.905395ms

Printing Unsolved Board!

Difficulty: **HARD**

Filled Cells: 36

```

-----
1 | _ _ _ | 8 _ 2 | _ 1 _ |
2 | 2 _ 9 | _ 1 4 | _ _ _ |
3 | 8 _ _ | _ 7 6 | 9 _ _ |
=====
4 | _ _ 1 | _ 2 _ | _ 9 _ |
5 | _ 4 6 | 9 3 _ | _ 7 _ |
6 | _ _ _ | _ 6 5 | _ 8 _ |
=====
7 | 9 6 8 | _ 4 3 | 1 2 _ |
8 | _ 3 _ | _ 8 _ | _ 6 _ |
9 | _ _ 7 | 6 _ 9 | _ 3 _ |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

18:08:45 [INFO] Generated Solution!

Printing Solved Board!

Difficulty: **HARD**

```

-----
1 | 6 5 4 | 8 9 2 | 3 1 7 |
2 | 2 7 9 | 3 1 4 | 6 5 8 |
3 | 8 1 3 | 5 7 6 | 9 4 2 |
=====
4 | 3 8 1 | 4 2 7 | 5 9 6 |
5 | 5 4 6 | 9 3 8 | 2 7 1 |
6 | 7 9 2 | 1 6 5 | 4 8 3 |
=====
7 | 9 6 8 | 7 4 3 | 1 2 5 |
8 | 4 3 5 | 2 8 1 | 7 6 9 |
9 | 1 2 7 | 6 5 9 | 8 3 4 |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

> play

18:09:11 [INFO] Writing 3 bytes to /dev/ttyUSB0

18:09:11 [INFO] Wrote 3 bytes!

18:09:11 [INFO] Bytes Read: [79, 75, 13, 10]

18:09:11 [INFO] Bytes Read: [68, 13, 10]

18:09:11 [INFO] Solved in: 78.342695ms

Difficulty: ULTRA
Filled Cells: 24

```

-----
1 | _ _ _ | _ _ 9 | _ 1 _ |
2 | _ _ _ | _ 2 _ | _ _ 6 |
3 | 2 _ _ | 3 _ 5 | _ _ _ |
=====
4 | _ _ _ | _ 7 _ | _ _ _ |
5 | 9 _ 7 | _ 6 _ | _ 5 _ |
6 | _ _ 3 | 8 _ _ | 7 _ _ |
=====
7 | _ _ 1 | _ _ _ | 3 6 8 |
8 | 3 _ 9 | _ _ _ | 4 _ _ |
9 | 7 _ _ | 2 _ _ | _ _ _ |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

17:45:07 [INFO] Generated Solution!
Printing Solved Board!
Difficulty: ULTRA

```

-----
1 | 8 3 5 | 6 4 9 | 2 1 7 |
2 | 1 9 4 | 7 2 8 | 5 3 6 |
3 | 2 7 6 | 3 1 5 | 9 8 4 |
=====
4 | 5 8 2 | 9 7 3 | 6 4 1 |
5 | 9 1 7 | 4 6 2 | 8 5 3 |
6 | 6 4 3 | 8 5 1 | 7 2 9 |
=====
7 | 4 2 1 | 5 9 7 | 3 6 8 |
8 | 3 5 9 | 1 8 6 | 4 7 2 |
9 | 7 6 8 | 2 3 4 | 1 9 5 |
-----
👉 | 1 2 3 | 4 5 6 | 7 8 9 |

```

> play

17:53:45 [INFO] Writing 3 bytes to /dev/ttyUSB0
17:53:45 [INFO] Wrote 3 bytes!
17:53:45 [INFO] Bytes Read: [79, 75, 13, 10]
17:53:59 [INFO] Bytes Read: [68, 13, 10]
17:53:59 [INFO] Solved in: 13.472012253s