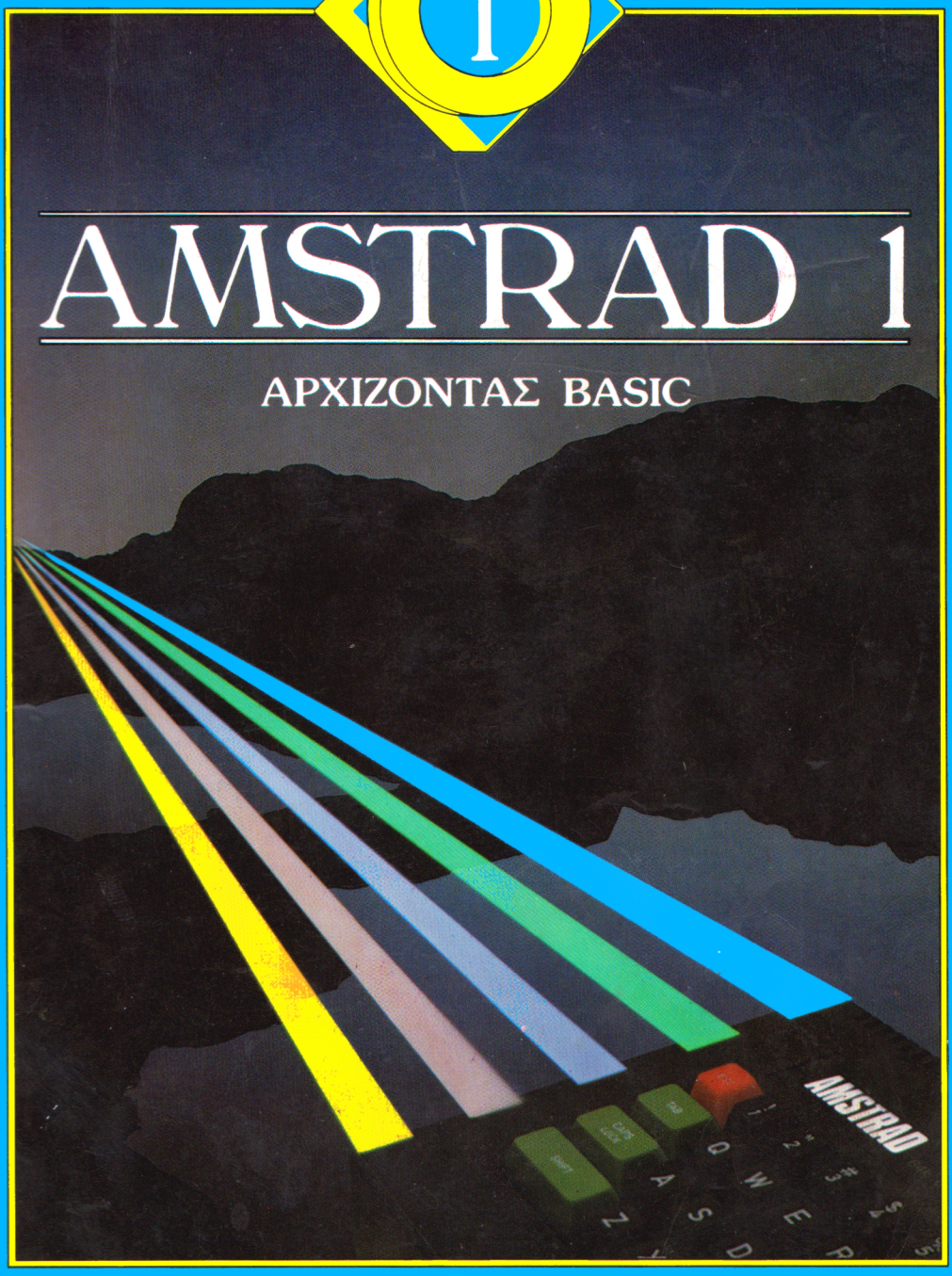
1

AMSTRAD 1

ΑΡΧΙΖΟΝΤΑΣ BASIC



Sean Gray & Eddy Maddix

ΕΛΛΗΝΙΚΗ ΕΚΔΟΣΗ

AMSTRAD
CPC 464, 6128
ΑΡΧΙΖΟΝΤΑΣ BASIC

ΒΙΒΛΙΟ 1

Sean Gray & Eddy Maddix

GLENTOP
PUBLISHERS■ LIMITED

Glentop Publishers Ltd,
Standfast House, Bath Place,
High Street, Barnet, Herts EN5 5XE
Tel: 01-441 4130

MPS
ΠΛΗΡΟΦΟΡΙΚΗ Β. ΕΛΛΑΔΟΣ
ΠΟΛΥΤΕΧΝΕΙΟΥ 47, ΤΗΛ. 540 246
546 25 - ΘΕΣΣΑΛΟΝΙΚΗ

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Αυτό είναι το πρώτο από μια σειρά δύο βιβλίων για τον Amstrad 464, 664, 6128. Ο σκοπός του βιβλίου αυτού είναι να διδάξει τον αναγνώστη τα βασικά της Basic του Amstrad, δηλαδή τις πιο κοινές εντολές και πως να τις χρησιμοποιήσετε. Επίσης ένας άλλος σκοπός είναι να επιδειχτούν διάφοροι τεχνικές προγραμματισμού περιλαμβανομένου και του δομημένου προγραμματισμού. Για να επιτευχθούν αυτοί οι σκοποί κάθε εντολή που αναπτύσσεται εξηγείται με προσεκτικά βήματα και στις περισσότερες περιπτώσεις αυτό περιλαμβάνει ένα πρόγραμμα επίδειξης που δείχνει στον αναγνώστη όχι μόνο τι κάνει η εντολή, αλλά και πως να χρησιμοποιηθεί σ' ένα πρόγραμμα.

Κατ' αρχάς ο αναγνώστης εισάγεται στις απλούστερες εντολές της Basic στο κεφάλαιο 1 και όταν έχει τελειώσει διαβάζοντας αυτό το βιβλίο είναι ικανός πλήρως να προγραμματίζει στη Basic του Amstrad.

Στο βιβλίο αυτό περιλαμβάνονται τρία παιχνίδια. Το πρώτο είναι ένα απλό παιχνίδι που μαντεύει αριθμούς, το δεύτερο είναι ένα παιχνίδι αναγραμματισμού και το τελευταίο είναι το παιχνίδι της κρεμάλας με τη χρήση των γραφικών του υπολογιστή. Ο αντικειμενικός σκοπός καθενός από τα παιχνίδια είναι να <u>φωτίσει</u> καθορισμένα σημεία. Π.χ. το παιχνίδι αριθμών οδηγεί τον αναγνώστη στον κόσμο των τυχαίων αριθμών <u>χρησιμοποιώντας</u> απλή λογική. Στο τέλος του κεφαλαίου 5 ο αναγνώστης θα μάθει διάφορους τρόπους αποθήκευσης πληροφοριών, πως να σχεδιάζει εικόνες και τα πλεονεκτήματα του modular δομημένου προγραμματισμού.

Σημειώνεται ότι, ως μια εισαγωγή των εντολών της BASIC του Amstrad το βιβλίο αυτό είναι αυτοδύναμο. Αυτοί που θέλουνε να μάθουν περισσότερα για τον Amstrad πρέπει ακόμη να διαβάσουν το βιβλίο δύο. Ελπίζουμε να κερδίσετε όσο το δυνατό περισσότερα διαβάζοντας αυτό το βιβλίο.

S. Gray

E. Maddix

Ιανουάριος 1985

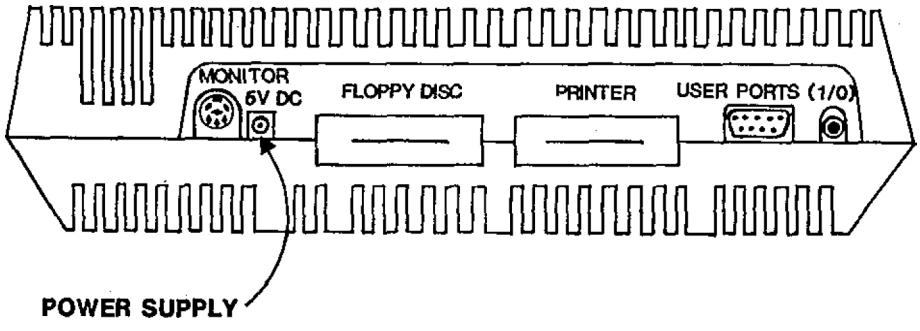
ΚΕΦΑΛΑΙΟ 1

ΜΕΡΟΣ ΠΡΩΤΟ

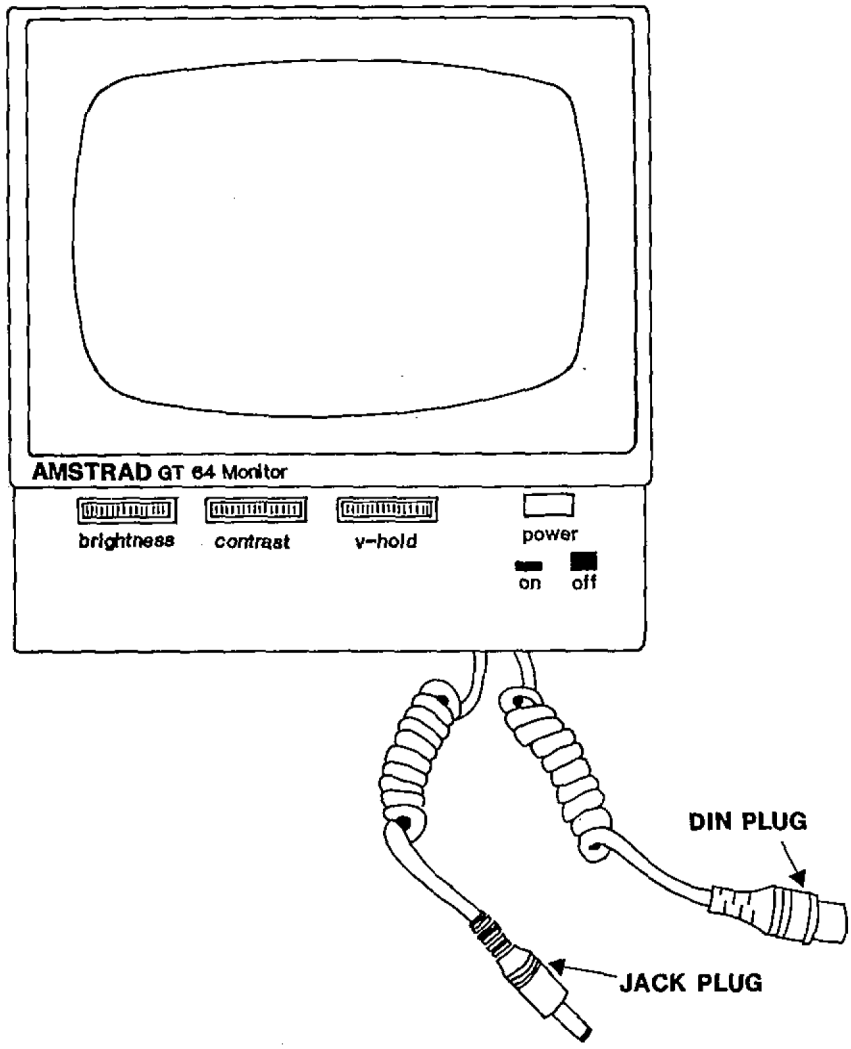
Κάθεστε άνετα; Τότε αρχίζουμε. Αυτό το μέρος του κεφαλαίου αναφέρεται στο πως τίθεται σε λειτουργία ο Amstrad σας. Σε αντίθεση με τους περισσότερους υπολογιστές, ο Amstrad έχει τη δική του, ειδικά γι' αυτόν σχεδιασμένη οθόνη. Επειδή η οθόνη είναι κατασκευασμένη για να δουλεύει με τον υπολογιστή Amstrad η εικόνα (και τα χρώματα) είναι πολύ καθαρότερα απ' ότι σε μία συνηθισμένη τηλεόραση.

1.1.1 Η Σύνδεση του Amstrad

Στην οθόνη του Amstrad βρίσκεται και ο μετασχηματιστής του υπολογιστή. Έτσι ελλατώνονται τα καλώδια που κρέμονται απο δώ κι' από κει. Στο μπροστινό τμήμα της οθόνης υπάρχουν δύο καλώδια, το ένα το μεγαλύτερο καταλήγει σε ένα βύσμα din και το άλλο σε ένα βύσμα τροφοδοσίας.



Εικόνα 1.1 (α) Το πληκτρολόγιο του Amstrad.

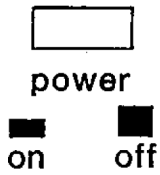


Εικόνα 1.1(β) Η οθόνη του Amstrad.

Το βύσμα τοποθετείται στην πρίζα στο πίσω μέρος τ οθόνης του Amstrad σύμφωνα με τις ενδείξεις που υπάρχο σ' αυτό (εικόνα 1.1(a)). Δίπλα στην πρίζα της οθόν υπάρχει ακόμη μία με την ένδειξη "5 VDC". Εκε τοποθετείται το βύσμα τροφοδοσίας. 'Οταν συνδεθο κατάλληλα και τα δύο είστε έτοιμοι ν' αρχίσετε.

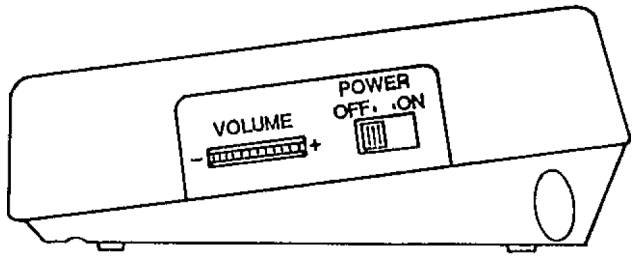
Στο μπροστά μέρος της οθόνης υπάρχει ένας διακόπτ με την ένδειξη POWER (εικόνα 1.2), πατώντας τον ανοίγε την οθόνη. Αυτό θέτει σε λειτουργία και τ

μετασχηματιστή του Amstrad αλλά όχι τον ίδιο τον υπολογιστή.



Εικόνα 1.2 Ο διακόπτης της οθόνης.

Ο υπολογιστής Amstrad έχει το δικό του διακόπτη ON/OFF που βρίσκεται στη δεξιά πλευρά του.



Εικόνα 1.3 Ο διακόπτης του πληκτρολογίου.

Αν είναι όλα τα καλώδια σωστά συνδεδεμένα και η οθόνη ανοικτή χτυπήστε ελαφρά το διακόπτη του πληκτρολογίου στο "ON". Τότε θ' ανάψει ένα κόκκινο φωτάκι στο πληκτρολόγιο. Ο προσωπικός σας υπολογιστής Amstrad είναι τώρα συνδεδεμένος και είναι έτοιμος ν' αρχίσει.

Μόλις τεθούν όλα σε λειτουργία θα δείτε το ακόλουθο μήνυμα:

Amstrad 64K Microcomputer (v1)
© 1984 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.

BASIC 1.0

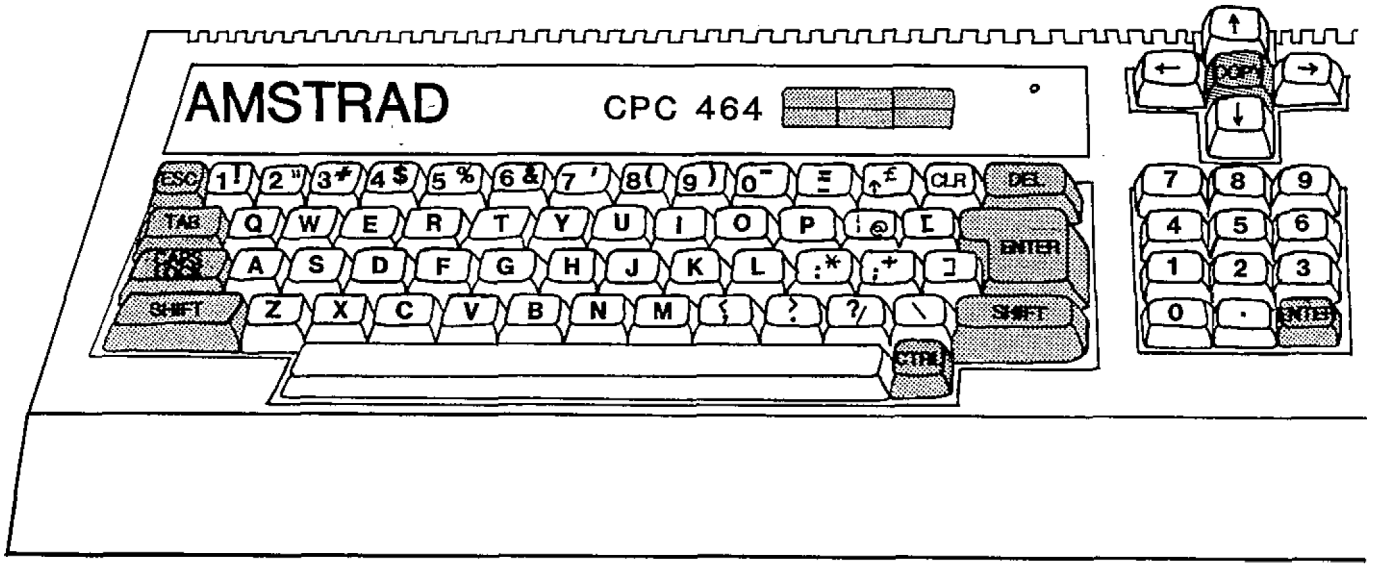
READY


Εικόνα 1.4 Η εμφάνιση της οθόνης αρχής.

Η πρώτη εμφάνιση μηνυμάτων μας δίνει διάφορες πληροφορίες. Η πρώτη σειρά μας λέει ποιον υπολογιστή έχουμε πάρει, στην προκειμένη περίπτωση ένα μικρουπολογιστή Amstrad με 64K. Ο υπολογιστής σας έχει 64K μνήμη, η οποία είναι πολλή. Η δεύτερη σειρά μας λέει ποιος σχεδίασε τον υπολογιστή Amstrad και σε ποιον ανήκουν τα συγγραφικά δικαιώματα του software δηλαδή εταιρείες Amstrad Consumer Electronics και Locomotive Software.

Μετά βλέπουμε το μήνυμα "Basic 1.0". Αυτό μας λέει ότι η γλώσσα που καταλαβαίνει ο Amstrad είναι η έκδοση 1.0 της BASIC. Υπάρχουν πολλές εκδόσεις της BASIC, σχεδόν μία για κάθε καινούριο υπολογιστή. Η λέξη BASIC προέρχεται από τα αρχικά των λέξεων Beginners All Purpose Symbolic Instruction Code.

Το μήνυμα Ready σημαίνει ότι ο υπολογιστής είναι έτοιμος ν' αρχίσει. Κάτω από το R του Ready υπάρχει ένα φωτεινό χρωματισμένο σημάδι, που ονομάζεται "ΔΡΟΜΕΑΣ". Με το δρομέα, ο υπολογιστής μας λέει, που θα εμφανιστεί ο επόμενος χαρακτήρας. Αν πιέσετε ένα από τα πλήκτρα των γραμμάτων στο πληκτρολόγιο, το αντίστοιχο γράμμα θα εμφανιστεί στην οθόνη εκεί όπου ήταν ο δρομέας. Ο δρομέας θα μετακινηθεί ένα διάστημα προς τα δεξιά. Αν πιέσετε ένα άλλο γράμμα τότε θα συμβεί πάλι το ίδιο. Αν συνεχίσετε να κρατάτε πατημένο ένα πλήκτρο ο δρομέας θα μετακινηθεί κατά μήκος της οθόνης μέχρι να φτάσει στη δεξιά πλευρά της. Μετά θα εμφανιστεί στην αριστερή πλευρά της οθόνης στην επόμενη σειρά.



Εικόνα 1.5 Το πληκτρολόγιο του Amstrad.

1.1.2 Το Πληκτρολόγιο

Ο Amstrad σας επιτρέπει να χρησιμοποιείτε το πληκτρολόγιο όπως θα χρησιμοποιούσατε τα πλήκτρα μιας γραφομηχανής. Αν πατήσετε ένα από τα πλήκτρα των γραμμάτων θα δείτε το αντίστοιχο γράμμα να εμφανίζεται σαν μικρός χαρακτήρας. Για να έχετε κεφαλαία γράμματα πατάτε απλά το πλήκτρο στην αριστερή πλευρά του πληκτρολογίου. Πατώντας μια φορά το πλήκτρο Caps Lock έχετε κεφαλαίους χαρακτήρες και κάθε γράμμα που πατάτε μετά απ' αυτό θα εμφανίζεται σαν κεφαλαίο.

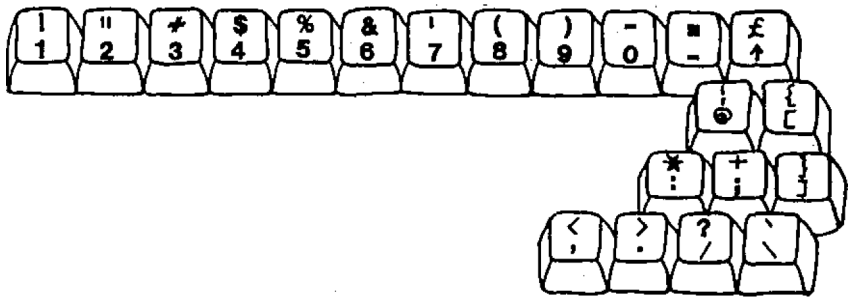
Για να ξαναγυρίσετε στους μικρούς χαρακτήρες πατάτε ακόμη μία φορά το πλήκτρο Caps Lock και έχετε τώρα μικρά γράμματα. Το πλήκτρο Caps Lock είναι γνωστό σαν ένας συνδετικός διακόπτης. Αν έχετε μικρά γράμματα και το πατήσετε πηγαίνετε στα κεφαλαία, αλλά αν είσασταν ήδη στα κεφαλαία πατώντας στο Caps Lock πηγαίνετε στα μικρά.

1.1.3 SHIFT

Στον Amstad υπάρχουν δύο SHIFT πλήκτρα. Είναι και τα δύο πράσινα και βρίσκονται το καθ' ένα σε μια από τις δύο πλευρές του πληκτρολογίου. Θα πρέπει να γίνει κατανοητό ότι δεν παίζει κανένα ρόλο το ποιο πλήκτρο SHIFT χρησιμοποιείτε. Και τα δύο κάνουν την ίδια δουλειά. Αν αυτό έχινε αντιληπτό τότε μπορούμε να δούμε τι κάνουν.

Θα προσέξατε ότι μερικά από τα πλήκτρα στο πληκτρολόγιο του Amstrad έχουν περισσότερα από ένα σύμβολα π.χ. το πλήκτρο 4 έχει επάνω του και το σύμβολο του δολλαρίου $. Για να έχετε το χαρακτήρα 4 πατάτε απλά το πλήκτρο. Κρατώντας πατημένο ένα από τα πλήκτρα SHIFT μπορείτε να έχετε τους επιπλέον χαρακτήρες που υπάρχουν πάνω σε μερικά από τα πλήκτρα. Οι "επιπλέον" χαρακτήρες ή χαρακτήρες που γράφονται κρατώντας κάτω το πλήκτρο SHIFT είναι αυτοί που βρίσκονται στο πάνω μέρος των πλήκτρων.

Τα παρακάτω πλήκτρα έχουν επιπλέον SHIFT-χαρακτήρες.

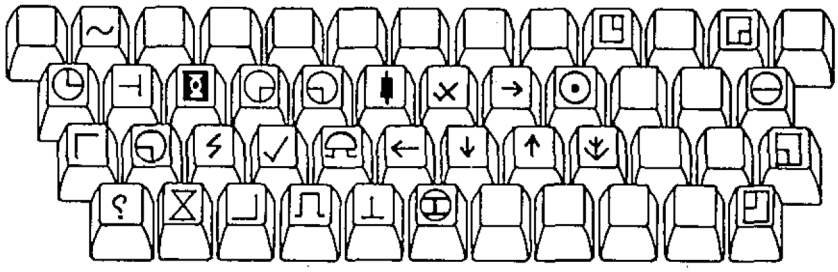


Εικόνα 1.6 Οι χαρακτήρες με shift.

Όλοι αυτοί SHIFT-χαρακτήρες μπορούν να ληφθούν κρατώντας κάτω ένα από τα πλήκτρα SHIFT και πατώντας το κατάλληλο πλήκτρο. Τα πλήκτρα των γραμμάτων δεν έχουν επιπλέον SHIFT-χαρακτήρες πάνω τους, αλλά όταν δε χρησιμοποιείται το CAPS LOCK, κρατώντας πατημένο το πλήκτρο SHIFT και πατώντας το πλήκτρο ενός γράμματος έχετε το κεφαλαίο γράμμα.

1.1.3.1 CTRL

Όπως μερικά πλήκτρα έχουν επιπλέον SHIFT-χαρακτήρες, άλλα έχουν ειδικές λειτουργίες ελέγχου. Αυτές οι λειτουργίες δεν απεικονίζονται πάνω στα πλήκτρα όπως μπορείτε να συμπεράνετε από την εικόνα 1.7. Για να έχετε αυτές τις ειδικές λειτουργίες θα πρέπει να κρατήσετε πατημένο το CTRL και να πατήσετε ταυτόχρονα το κατάλληλο πλήκτρο.



Εικόνα 1.7 Τα ειδικά πλήκτρα CTRL.

Υπάρχει ένα CTRL – πλήκτρο που δε φαίνεται στην εικόνα 1.7 και είναι το CTRL και το P. Κρατώντας κάτω το CTRL και πατώντας το πλήκτρο "P" δε θα εμφανιστεί ένας

χαρακτήρας στην οθόνη αλλά θα ακουστεί ένα "μπιπ". Αν δεν το ακούτε, γυρίστε το ποτενσιόμετρο της έντασης που υπάρχει κοντά στο διακόπτη με τον οποίο ανοίγει ή κλείνει το πληκτρολόγιο και πατήστε το CTRL και το P ξανά. Αυτή τη φορά (αν γυρίσατε τον τροχό, προς τη σωστή κατεύθυνση) θ' ακούσετε το σύντομο ήχο, που είναι, στον υπολογιστή το αντίστοιχο με το κουδούνι μιας κοινής γραφομηχανής.

1.1.4 Αυτόματη επανάληψη

Τα περισσότερα πλήκτρα στο κύριο τμήμα του πληκτρολογίου έχουν "αυτόματη επανάληψη". Αυτό σημαίνει ότι αν κρατήσετε πατημένο το πλήκτρο για περισσότερο από μισό δευτερόλεπτο ο χαρακτήρας που πατάτε θ' αρχίσει να εμφανίζεται κατ' επανάληψη. Για να το διαπιστώσετε κρατήστε πατημένο το πλήκτρο Η. Διατηρώντας το πατημένο μετά από έξι και μισή σειρές Η, ο Amstrad θα δώσει ένα δυνατό μπιπ και ο δρομέας θα παραμένει στο ίδιο σημείο. Αυτός ο ήχος σας λέει ότι ο δρομέας έχει προχωρήσει όσο ήταν δυνατό και θα συνεχίσει να ακούγεται μέχρι να απομακρύνετε το δάκτυλό σας από το πλήκτρο.

1.1.4.1 DEL

Αν φτάσουμε στο σημείο, στο οποίο να μη μπορούμε να πατήσουμε κάποιο από τα πλήκτρα χαρακτήρων, χωρίς να ακουστεί μπιπ, θα πρέπει να γνωρίζουμε τη λειτουργία του πλήκτρου DEL. Το πλήκτρο DEL είναι πράσινο και βρίσκεται αριστερά επάνω στο πληκτρολόγιο σας. Όταν είναι πατημένο ο χαρακτήρας που βρίσκεται πριν από το δρομέα σβήνεται. Πατώντας το πλήκτρο DEL μπορούμε να σβήσουμε όλα τα Η που τυπώθηκαν. Και το DEL έχει αυτόματη επανάληψη αν το κρατήσετε πατημένο.

Μόλις σβηστεί το τελευταίο Η θ' ακούσετε ξανά το μπιπ. Μ' αυτόν τον τρόπο ο υπολογιστής σας λέει ότι δε μπορείτε να σβήσετε πέρα από την αρχή της σειράς. Μόλις σηκώσετε το δάκτυλό σας από το DEL το μπιπ θα σταματήσει.

1.1.5 Διαστήματα

Κατά μήκος της κάτω πλευράς του πληκτρολογίου υπάρχει ένα μακρύ γκρι πλήκτρο. Όταν πατήσετε αυτό το πλήκτρο ο δρομέας θα κινηθεί προς τα δεξιά αλλά δε θα εμφανιστεί τίποτε στην οθόνη. Μ' αυτό το πλήκτρο που είναι γνωστό σαν "μπάρα διαστημάτων" παίρνετε ένα χαρακτήρα διαστήματος (κενό διάστημα μεταξύ δύο χαρακτήρων - κενός χαρακτήρας). Επειδή δε μπορείτε να δείτε αυτόν το χαρακτήρα, στο βιβλίο αυτό θα το συμβολίζουμε με το σύμβολο δέλτα Δ. Αυτό χρησιμοποιείται για να δηλωθεί ότι υπάρχει ένα κενό σε περιπτώσεις στις

οποίες αυτά είναι πολύ σημαντικά. Έτσι όταν βλέπετε αυτό το σύμβολο να θυμάστε να πατάτε τη μπάρα διαστημάτων.

Εναλλακτικά, αν απαιτούνται πολλά κενά θα δείτε κάτι όπως αυτό: <12 κενά>. Αυτό σημαίνει ότι εσείς θα πρέπει να πιέσετε την μπάρα διαστημάτων δώδεκα φορές.

1.1.6 Αρχίζοντας στη BASIC

Τώρα που ξέρετε τι κάνουν τα περισσότερα πλήκτρα είναι καιρός να μάθετε τι μπορεί να κάνει ο καινούργιος σας υπολογιστής Amstrad. Εδώ αρχίζετε να μαθαίνετε πραγματικά.

Ενώ πατούσατε τα πλήκτρα για να δείτε τι κάνουν, μπορεί να έχετε δει σε μερικές περιπτώσεις να εμφανίζεται στην οθόνη το ακόλουθο μήνυμα:

Syntax error

Αυτό είναι ένα πολύ σημαντικό μήνυμα για σας από τον υπολογιστή σας. Ο Amstrad σας λέει ότι δεν καταλαβαίνει τι πληκτρολογήσατε. Syntax error σημαίνει ότι ο υπολογιστής δεν καταλαβαίνει τον τρόπο με τον οποίο προσπαθείτε να του πείτε κάτι.

Ο Amstrad, όπως οι περισσότεροι οικιακοί υπολογιστές, καταλαβαίνει τη γλώσσα των υπολογιστών που ονομάζεται BASIC. Η γλώσσα αυτή αναπτύχθηκε ειδικά για να κάνει δυνατό έναν εύκολο διάλογο ανάμεσα στον υπολογιστή και στο χρήστη. Για να περιγράψουμε τις διαφορές ανάμεσα στη BASIC και τα Αγγλικά πληκτρολογήστε το εξής:

SAY HELLO

Αφού γράψετε αυτό ο δείκτης θα τοποθετηθεί μετά το Ο στο HELLO. Τι κάνουμε τώρα;

1.1.6.1 ENTER

Το πλήκτρο ENTER είναι δικαιολογημένα το πιο σημαντικό πλήκτρο στο πληκτρολόγιο του Amstrad. Αυτό που κάνει το πλήκτρο αυτό είναι να λέει στον υπολογιστή ότι εσείς έχετε τελειώσει την πληκτρολόγηση των οδηγιών σας και θέλετε να τις δεχτεί ο υπολογιστής. **Όταν τελειώνετε την πληκτρολόγηση των οδηγιών σας πρέπει πάντα να θυμάστε να πατάτε το πλήκτρο ENTER.**

Στο παρακάτω παράδειγμα μόλις τελειώσετε την πληκτρολόγηση μιας οδηγίας πατήστε το πλήκτρο ENTER. Θα πάρετε το ακόλουθο μήνυμα:

Syntax error

Παρόλο που εμείς δώσαμε στον υπολογιστή μια καθαρή οδηγία SAY HELLO, αυτή είναι μια οδηγία που δεν την καταλαβαίνει ο υπολογιστής. Για να πείτε στον υπολογιστή να πει HELLO πρέπει να χρησιμοποιήσετε την εντολή της BASIC, PRINT.

1.1.6.2 PRINT

Η PRINT είναι η εντολή με την οποία λέτε στον υπολογιστή ότι θέλετε να σας πει κάτι. Για να πείτε στον Amstrad να σας πει HELLO με έναν τρόπο που να τον καταλαβαίνει, πρέπει να πληκτρολογήσετε την ακόλουθη εντολή:

PRINT "HELLO"

Τώρα αν πατήσετε το ENTER ο υπολογιστής θα δείξει στην οθόνη:

HELLO

Μπράβο σας! Κάνατε τον υπολογιστή να σας καταλάβει. Η εντολή PRINT λέει στον υπολογιστή να εμφανίσει στην οθόνη οτιδήποτε βλέπει μέσα σε εισαγωγικά. Σ' αυτό το παράδειγμα βλέπει τη λέξη "HELLO": και αυτό εμφανίστηκε αμέσως στην οθόνη. Κάτω απ' αυτό θα δείτε το μήνημα:

Ready

Αυτό είναι η παρότρυνση Ready, μ' αυτό σας λέει ότι σας έχει καταλάβει και έχει υπακούσει στις οδηγίες σας χωρίς κανένα πρόβλημα. Ο δρομέας περιμένει υπομονετικά να πληκτρολογήσετε κάτι.

Τα διπλά εισαγωγικά σε κάθε πλευρά του HELLO είναι πολύ σημαντικά γιατί λένε στον υπολογιστή να τυπώσει ακριβώς ότι περικλείεται ανάμεσα σ' αυτά. Οτιδήποτε βρίσκεται μέσα στα εισαγωγικά θα τυπωθεί ακόμη και αν είναι αγενές, ανορθόγραφο ή λογικά λανθασμένο.

```
PRINT "Amstrad is rubbish"
```

Αυτό δεν είναι σωστό αλλά εσείς είπατε στον Amstrad να το τυπώσει και αυτός θα το κάνει.

1.1.6.3 Σειρές χαρακτήρων

Οι χαρακτήρες που βρίσκονται μέσα σε δύο εισαγωγικά ονομάζονται σειρές χαρακτήρων. Για τον Amstrad κάθετι που περικλείεται σε εισαγωγικά θεωρείται σαν μια σειρά από χαρακτήρες.

Η εντολή PRINT μπορεί επίσης να χρησιμοποιηθεί για την εμφάνιση αριθμών. Για να αποδειχθεί αυτό το σημείο πληκτρολογήστε:

```
PRINT 8
```

Πατώντας το πλήκτρο ENTER θα δείτε να εμφανίζεται στην οθόνη ένα 8. Η διαφορά στην εκτύπωση αριθμών και σειρών χαρακτήρων είναι ότι, οι αριθμοί δε χρειάζονται εισαγωγικά. Αν το 8 ήταν μέσα σε εισαγωγικά, π.χ.

```
PRINT "8"
```

Το 8 θα φυλασσόταν σαν σειρά χαρακτήρων, παρόλο που η τελευταία εμφάνιση θα ήταν η ίδια με την εκτύπωση του 8. Σημειωτέον ότι στην περίπτωση του αριθμού 8, ο υπολογιστής έχει βάλει ένα διάστημα πριν απ' αυτό, δεν έκανε όμως το ίδιο με το "8".

Η εντολή PRINT μπορεί να εκτελεί υπολογισμούς και να εμφανίζει την απάντηση. Αν πληκτρολογήσετε PRINT και μετά μια αριθμητική πράξη και πατήσετε το ENTER, θα δείτε την απάντηση ("/" σημαίνει διαίρεση, "*" σημαίνει πολλαπλασιασμός) π.χ.

```
PRINT 47/5
 9.4
Ready
```

Χρησιμοποιώντας το PRINT κατ' αυτόν τον τρόπο εύκολο να γίνουν χρήσιμοι υπολογισμοί.

1.1.6.4 LET

Όλα ότι έχουμε κάνει μέχρι τώρα είναι να λέμε στον υπολογιστή να εμφανίσει διάφορα πράγματα στην οθόνη. Το επόμενο βήμα είναι του πούμε να θυμάται κάτι. Αυτό γίνεται χρησιμοποιώντας την εντολή LET.

Η εντολή LET λέει στον υπολογιστή να θυμάται ότι θέλουμε να του πούμε. Όταν χρησιμοποιούμε την εντολή αυτή, χρειάζεται να του λέμε δύο πράγματα. Το πρώτο είναι με ποιο όνομα θα θυμάται και το δεύτερο τι θέλουμε εμείς να θυμάται.

1.1.7 Μεταβλητές

Όταν ζητούμε από τον Amstrad να θυμηθεί ένα γεγονός πρέπει να ορίζουμε ειδικά ένα όνομα με το οποίο θα θυμάται αυτό το γεγονός. Αυτό το όνομα λέγεται μεταβλητή. Ονομάζεται μεταβλητή διότι η τιμή αυτού που θυμάται ο υπολογιστής μπορεί κάθε φορά να αλλάζει όπως συμβαίνει με μια μεταβλητή.

Η ονομασία των μεταβλητών ελέγχεται από κανόνες οι οποίοι ευτυχώς είναι πολλοί.

Ο πρώτος κανόνας είναι ότι οι μεταβλητές πρέπει ν' αρχίζουν με ένα γράμμα π.χ. A7.

Ο δεύτερος είναι ότι το όνομα μιας μεταβλητής δεν πρέπει να είναι μεγαλύτερο από σαράντα χαρακτήρες σε μήκος.

Ο τρίτος είναι ότι το όνομα της μεταβλητής δεν θα πρέπει να περιέχει σημεία στίξης.

Ο τέταρτος και τελευταίος κανόνας είναι ότι οι μεταβλητές που χρησιμοποιούνται για να αποθηκεύονται σειρές χαρακτήρων πρέπει να έχουν το σύμβολο του δολλαρίου ($) σαν τελευταίου χαρακτήρα του ονόματος. Αυτό δε χρειάζεται σε μεταβλητές που χρησιμοποιούνται για να αποθηκεύονται αριθμητικά στοιχεία.

Μεταβλητές που έχουν περιεχόμενο σειρές χαρακτήρων ονομάζονται μεταβλητές σειρών χαρακτήρων. Ο υπολογιστής μπορεί να καταλάβει τη διαφορά ανάμεσα σε μεταβλητές σειρών χαρακτήρων και σε αριθμητικές μεταβλητές δηλαδή μεταβλητές που χρησιμοποιούνται για να αποθηκεύονται αριθμοί, διότι οι πρώτες λήγουν σε ένα $ ενώ οι δεύτερες όχι.

Έτσι για να πείτε στον υπολογιστή να θυμηθεί το όνομά σας πληκτρολογήστε τα ακόλουθα και μετά πατήστε το ENTER.

```
LET NAME$="ALBERT"
```

(Αν το όνομα σας δεν είναι ALBERT θα μπορούσατε να πληκτρολογήσετε το δικό σας).

Μόλις πατήσετε το ENTER θα δείτε το μήνυμα Ready και θα εμφανιστεί ο δρομέας. Δεν θα φαίνεται να έχουν γίνει πολλά, αλλά μέσα στον Amstrad θα έχει δοθεί μια περιοχή της μνήμης στο όνομα NAME$ και θα έχει τοποθετηθεί μέσα σ' αυτή η σειρά χαρακτήρων ALBERT.

Η ανάγνωση και μετάφραση της εντολής LET γίνεται ως εξής:

i. Ο Amstrad διαβάζει LET NAME$ και το μεταφράζει σε: "πάρε μία περιοχή μνήμης και ονόμασέ την NAME$ Βλ. Εικ. 1.8

Εικόνα 1.8 Μια περιοχή μνήμης ονομασμένη NAME$.

ii. Το επόμενο που διαβάζει είναι το τμήμα ="ALBERT" και το μεταφράζει σε "αποθήκευσε" τη σειρά χαρακτήρων ALBERT στη μεταβλητή χαρακτήρων NAME$. Βλ. Εικ. 1.9

Εικόνα 1.9 Αποθήκευση του ALBERT στη NAME$.

Ο Amstrad προσέχει επίσης ότι η μεταβλητή έχει το σύμβολο του δολλαρίου στο τέλος του ονόματος ΚΑΙ ότι η έκφραση (τι είναι να γίνει) περικλείεται σε εισαγωγικά. Αν η μεταβλητή είναι μία μεταβλητή σειράς χαρακτήρων, αλλά η έκφραση δεν είναι μέσα σε εισαγωγικά τότε θα δείτε το παρακάτω μήνυμα:

```
Type mismatch
```

Αυτό το μήνυμα θα δοθεί επίσης, αν η μεταβλητή είναι αριθμητική αλλά η έκφραση βρίσκεται μέσα σε εισαγωγικά. Το μήνυμα "Type mismatch" είναι ο τρόπος με τον οποίο ο υπολογιστής μας λέει ότι έγινε μια απόπειρα να τοποθετηθεί μια σειρά χαρακτήρων σε μια αριθμητική περιοχή ή ένα αριθμητικό στοιχείο σε μια περιοχή σειράς χαρακτήρων.

Έχοντας διαβάσει την τελευταία σελίδα, ίσως έχετε ξεχάσει τι βάλατε (ορίσατε) τη μεταβλητή NAME$. Για να φρεσκάρετε τη μνήμη σας πληκτρολογήστε το εξής:

```
PRINT NAME$
```

Ο υπολογιστής θα το ερμηνεύσει σαν "να βρεθεί η περιοχή μνήμης που ονομάζεται NAME$ και να εκτυπωθεί, δηλαδή να εμφανιστεί στην οθόνη οτιδήποτε υπάρχει μέσα σ' αυτήν την περιοχή".

Το LET δουλεύει με αριθμούς κατά τον ίδιο τρόπο.

```
LET AGE=2
```

Εδώ τοποθετήθηκε η τιμή σε μια αριθμητική μεταβλητή που ονομάζεται AGE.

Η εντολή LET μπορεί επίσης να χρησιμοποιηθεί για να εκτελεί αθροίσεις.

```
LET AGE=AGE+1
```

Αυτό λέει στον υπολογιστή να προσθέσει 1 στην τιμή που είναι αποθηκευμένη στη μεταβλητή AGE και να αποθηκεύσει το αποτέλεσμα πάλι στη μεταβλητή AGE. Αυτό θ' αντικαταστήσει ότι υπήρχε αρχικά εκεί. Για να το ελέγξετε αυτό πληκτρολογήστε:

```
PRINT AGE
```

Η απάντηση που θα πάρετε είναι 3.

1.1.8 Το πρώτο σας πρόγραμμα: Αριθμοί εντολών και RUN

Μέχρι τώρα όλες οι εντολές που έχουν εισαχθεί στον υπολογιστή γίνονται (ή εκτελούνται) αμέσως αφού πατηθεί το πλήκτρο ENTER. Όλα αυτά είναι γνωστά σαν ΑΜΕΣΗ ΕΙΣΑΓΩΓΗ ή σαν ΕΝΤΟΛΕΣ ΑΜΕΣΗΣ ΜΟΡΦΗΣ. Μόλις εκτελεστούν μια φορά δεν μπορούν να ξαναεκτελεστούν, και χάνονται.

Παρ' όλα αυτά, όταν χρειάζονται προγράμματα, βρίσκονται αποθηκευμένα στη μνήμη και επαναδραστηριοποιούνται όταν απαιτείται. Αυτό που διακρίνει μια άμεση εντολή, από μια εντολή σε ένα πρόγραμμα, είναι ό,τι το πρόγραμμα έχει αριθμούς εντολών. Οπότε πατιέται το πλήκτρο ENTER, ο Amstrad βλέπει τα εισαγόμενα και αν δει έναν αριθμό στην αρχή της εντολής, αποθηκεύει το εισαγόμενο στη μνήμη σαν μία σειρά προγράμματος. Έτσι, αν οι πρώτες άμεσες εντολές επανατοποθετηθούν στο πρόγραμμα 1.1, θα αποτελέσουν ένα σύντομο όπως είναι πραγματικό πρόγραμμα. Κάθε σειρά ενός προγράμματος στη γλώσσα Basic αναφέρεται σε μια εντολή. Η διαφορά ανάμεσα στις δύο είναι ότι οι άμεσες εντολές εισάγονται κατ' ευθείαν ενώ οι εντολές ενός προγράμματος από τμήμα ενός προγράμματος. Όταν πληκτρολογείται PRINT NAME$ σε άμεση μορφή αυτό είναι μια άμεση εντολή. Όταν πληκτρολογείται 30 PRINT NAMES$ αυτό γίνεται κατανοητό σαν εντολή προγράμματος.

Ποιος είναι ο αριθμός μιας εντολής, στην πραγματικότητα δεν παίζει μεγάλο ρόλο, εφόσον αυτός είναι ένας ακέραιος θετικός αριθμός μεταξύ του 1 και του 65536.

Εκείνο που είναι σημαντικό είναι η κατάταξη των αριθμών των εντολών διότι ο Amstrad θ' αρχίσει το πρόγραμμα από τον μικρότερο αριθμό εντολής και θα προχωρήσει με αυξανόμενο αριθμό σειράς εκτός, αν του πούμε να κάνει διαφορετικά. Πληκτρολογήστε το παρακάτω πρόγραμμα (θυμηθείτε να πατάτε το ENTER στο τέλος κάθε νέας σειράς!):

ΠΡΟΓΡΑΜΜΑ 1.1

```
20 LET NAME$="ALBERT"
30 PRINT NAME$
```

Αυτή τη φορά, όταν πατηθεί το ENTER η μηχανή απαντά μετακινώντας το δείκτη μία σειρά πιο κάτω, δεν έχουν εκτελεστεί οι σειρές του προγράμματος. Για να εκτελεστούν, πληκτρολογήστε απλά RUN. Μόλις πατήσετε το ENTER το πρόγραμμα θα τυπώσει στην οθόνη τη λέξη ALBERT. Μόλις το κάνετε αυτό, θα έχετε τρέξει το πρώτο σας πραγματικό πρόγραμμα.

1.1.8.1 INPUT

Στο πρόγραμμα 1.1, το όνομά σας αποθηκεύτηκε άμεσα στο πρόγραμμα, πράγμα που σημαίνει βεβαίως ότι το πρόγραμμα είναι χία να χρησιμοποιηθεί μόνον από σας. Για να γίνει πιο χρήσιμο πρέπει να είναι εύκολο να μπορεί κανείς να ορίσει την τιμή της NAME$ αφού αρχίσει να τρέχει το πρόγραμμα. Αυτό μπορεί να γίνει χρησιμοποιώντας την εντολή INPUT η οποία κάνει το πρόγραμμα να σταματά και να περιμένει μέχρι να εισάγει, ο χρήστης την απαιτούμενη πληροφορία μέσω του πληκτρολογίου. Ο υπολογιστής χρειάζεται επίσης να ξέρει ποιο όνομα μεταβλητής να δώσει σ' αυτή την πληροφορία. Το πρόγραμμα 1.2 δείχνει μια τιμή στη NAME$ στην αρχή του προγράμματος. Για να σβηστεί η παλιά σειρά 20 χρειάζεται μόνο να πληκτρολογηθεί η καινούρια και να πατηθεί μετά το ENTER. Ο υπολογιστής τότε θα γράφει τη νέα σειρά πάνω στην παλιά. Για να εμφανιστεί το τροποποιημένο πρόγραμμα στην οθόνη, πληκτρολογήστε LIST και πατήστε ENTER.

ΠΡΟΓΡΑΜΜΑ 1.2

```
20 INPUT NAME$
30 PRINT NAME$
```

'Οταν τρέξει το πρόγραμμα 1.2 θα εμφανιστεί ένα ερωτηματικό, που υποδεικνύει ότι ο υπολογιστής περιμένει μια εισαγωγή πληροφοριών. Πληκτρολογώντας το όνομά σας και πατώντας το ENTER, ο υπολογιστής θα καταχωρήσει το INPUT στη NAME$, δηλαδή θα βάλει το όνομά σας στη μεταβλητή NAME$ και στη σειρά 30 και θα εκτυπώσει την τιμή της NAME$ - το όνομα σας.

Το INPUT μπορεί επίσης να χρησιμοποιηθεί για να εισαχθούν αριθμοί. Η μόνη διαφορά είναι ότι αυτοί πρέπει να τοποθετηθούν σε μια αριθμητική περιοχή, δηλαδή

```
2 INPUT AGE
```

Το να πληκτρολογηθεί ένας αριθμός σε μία μεταβλητή σειράς χαρακτήρων θα σημαίνει ότι ο αριθμός θα αποθηκευτεί σαν σειρά χαρακτήρων. Παρ' όλα αυτά, δεν μπορείτε να πληκτρολογήσετε ένα γράμμα σε μία αριθμητική μεταβλητή. Αν προσπαθήσετε, αφού πατήσετε το ENTER θα πάρετε το ακόλουθο μήνυμα:

```
?Redo from start
```

Μ' αυτόν τον τρόπο σας λέει ο υπολογιστής ότι η εισαγωγή πληροφοριών δεν ήταν του τύπου που περίμενε. Η παρότρυνση με το ερωτηματικό θα εξαφανιστεί και θα πρέπει τώρα να πληκτρολογήσετε έναν αριθμό.

Αφού αποθηκευτεί ένα τμήμα ενός προγράμματος στη μνήμη, θα χρειαστεί να προστεθούν καινούριες εντολές και ευτυχώς ο Amstrad το χειρίζεται αυτό πολύ εύκολα: εμφανίζει στην οθόνη τις εντολές καθώς εισάγονται. Παρ' όλα αυτά ας υποθέσουμε ότι θέλουμε να ξαναδούμε τι κάνει το πρόγραμμα στη σειρά 10, όταν εμείς βρισκόμαστε στη σειρά.5000.

1.1.8.2 LIST

Υπάρχει μια εντολή στην BASIC που ονομάζεται LIST. Αυτή η εντολή λέει στον υπολογιστή να εμφανίσει όλες τις σειρές του προγράμματος μας στην οθόνη σε αριθμητική διάταξη. Δοκιμάστε να πληκτρολογήσετε το LIST τώρα. Θα βλέπατε αφού έχετε πατήσει το ENTER το πρόγραμμά σας να περνάει από την οθόνη. Παρ' όλα αυτά η εντολή LIST του Amstrad είναι ακόμα εύχρηστη. Αν πληκτρολογήσετε LIST 30 για παράδειγμα, θα δείτε στην οθόνη τη σειρά 30. Οι διάφορες μορφές της εντολής LIST δίνονται παρακάτω:

LIST	εμφανίζει όλο το πρόγραμμα
LIST 30	εμφανίζει μόνο την εντολή 30
LIST-30	εμφανίζει το πρόγραμμα μέχρι και την εντολή 30
LIST 30-	εμφανίζει το πρόγραμμα από την εντολή 30
LIST 30-60	εμφανίζει τις εντολές 30 μέχρι και 60.

1.1.9 Μορφοποιημένες εμφανίσεις στην οθόνη – Διαχωριστές

Μέχρι εδώ, όλες οι εντολές PRINT που έχετε χρησιμοποιήσει είχαν να εκτυπώσουν κάτι απλό. Παρ' όλα αυτά είναι μερικές φορές αναγκαίο να εκτυπωθούν διάφορα ξεχωριστά πράγματα στην οθόνη ταυτόχρονα. Αυτό γίνεται στη BASIC με τη μορφή χαρακτηριστικών που λένε στον υπολογιστή ποια μορφή εμφάνισης απαιτείται στην οθόνη. Έτσι, για να τυπωθεί η NAME$, στο πρόγραμμα 1.2, στην οθόνη δύο φορές, θα ήταν δυνατό να γραφτεί μια σειρά σε ένα πρόγραμμα που έχει μια εντολή PRINT που ακολουθείται από δύο NAME$. Παρ' όλα αυτά, ο Amstrad περιμένει να δει χωριστές μεταβλητές σειρών χαρακτήρων σε εντολές PRINT. Έτσι λοιπόν τα σύμβολα που χρησιμοποιούνται για να τις χωρίζουν είναι γνωστά σαν "διαχωριστές".

Για να δοκιμάσετε αυτό, μπορείτε να τροποποιήσετε το πρόγραμμα 1.2 για να εκτυπώσει την NAME$ τέσσερεις φορές, χρησιμοποιώντας πρώτα απ' όλα το διαχωριστή "," (κόμμα). Αλλάξτε την εντολή 30 όπως παρακάτω:

```
30 PRINT NAME$,NAME$,NAME$,NAME$
```

Το να πληκτρολογήσετε αριθμό εντολής χωρίς τίποτε, λέει στον υπολογιστή να ψάξει μέχρι να βρει εκείνη την εντολή στη μνήμη του. Αφού τή βρει σβήνει οτιδήποτε βρισκόταν σ' αυτήν και ξεκινάει πρόθυμα τον αριθμό εντολής.

Αν πληκτρολογήσετε έναν αριθμό εντολής που δεν υπάρχει π.χ. 12 και μετά πατήσετε το ENTER, ο Amstrad θα ψάξει στη μνήμη του μέχρι να καταλάβει ότι δεν υπάρχει καμμία σειρά 12 για να ξεκάσει. Το ακόλουθο μήνυμα λάθους εμφανίζεται:

```
Line does not exist
```

λέγοντας σας ότι έχετε κάνει λάθος και η εντολή δεν υπάρχει.

Τώρα σβήστε την εντολή 2 πληκτρολογώντας το 2 και πατώντας το ENTER. Θα έχετε τώρα το πρόγραμμα 1.2(a)

ΠΡΟΓΡΑΜΜΑ 1.2(a)

```
20 INPUT NAME$
30 PRINT NAME$,NAME$,NAME$,NAME$
```

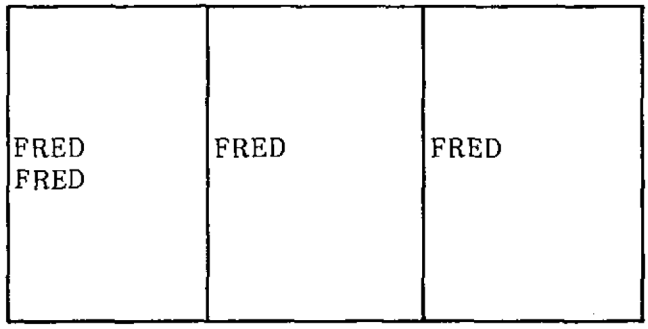
Όταν αυτό τρέξει χρησιμοποιωντας τη λέξη "FRED" σαν εισαγόμενο στοιχείο για τη μεταβλητή NAME$ θα φέρει σαν αποτέλεσμα την παρακάτω εμφάνιση στην οθόνη.

```
FRED                FRED                FRED
FRED
```

Εικόνα 1.10 Εμφάνιση οθόνης με διαχωριστή το ,

Σε κάθε μία από τις μεταβλητές σειρών χαρακτήρων που εκτυπώνεται στην οθόνη είναι κατανεμημένο το ένα τρίτο της οθόνης. Αυτό είναι γνωστό σαν "**πεδίο εκτύπωσης**" και δίνει ένα χρήσιμο μέσο κατανομής σε στήλες μιας εμφάνισης μεταβλητών σειρών χαρακτήρων ή αριθμητικών μεταβλητών.

Ένα αξιοσημείωτο στοιχείο για τις μεταβλητές σειρών είναι, ότι αν αυτές είναι μεγαλύτερες από 12 χαρακτήρες, θα χεμιάσουν το πεδίο εκτύπωσης και θα περάσουν στο επόμενο. Σ' αυτή την περίπτωση, ο επόμενος χαρακτήρας που είναι να εκτυπωθεί μετά από ένα διαχωριστή κόμμα θα μετατοπιστεί στο επόμενο διαθέσιμο πεδίο. Βλ. Εικ. 1.11



Εικόνα 1.11 Παράδειγμα πεδίων της PRINT.

'Ενας άλλος διαχωριστής είναι η άνω τελεία ;, που έχει σαν αποτέλεσμα να εκτυπώνεται μεταβλητή σειράς χαρακτήρων αμέσως μετά από μία άλλη. Αυτό φαίνεται στο πρόγραμμα 1.2(b) όπου η σειρά 30 έχει επεκταθεί για να συμπεριλάβει και ένα διαχωριστή ;.

ΠΡΟΓΡΑΜΜΑ 1.2(b)

```
20 INPUT NAME$
30 PRINT NAME$,NAME$,NAME$,NAME$;MANE$
```

Αυτό δείχνει το αποτέλεσμα που προκαλεί η χρήση της άνω τελείας ; που δίνει την εμφάνιση που φαίνεται στην εικόνα 1.12.

```
FRED                    FRED                    FRED
FREDFRED
```

Εικόνα 1.12 Εμφάνιση οθόνης με διαχωριστή το , και ;.

Σε όλα τα παραδείγματα μέχρι εδώ, οι διαχωριστές χρησιμοποιήθηκαν με μεταβλητές σειρών χαρακτήρων. Είναι αυτονόητο ότι η χρήση τους με αριθμητικές μεταβλητές είναι σχεδόν ταυτόσημη. Η μοναδική διαφορά είναι ότι οι αριθμοί εκτυπώνονται με ένα διάστημα σε κάθε μια πλευρά τους. 'Ετσι μπορούν να εκτυπωθούν εντολές που περιέχουν συνδυασμούς μεταβλητών σειρών χαρακτήρων και αριθμητικών μεταβλητών που διαχωρίζονται με κόμμα ή άνω τελεία.

1.1.10 Μηνύματα

Είδαμε ότι η εντολή INPUT μπορεί να χρησιμοποιηθεί σε ένα πρόγραμμα για να πάρει μία απάντηση από το χρήστη. Δυστυχώς, το ερωτηματικό δε βοηθάει πολύ για να ζητήσει κανείς πληροφορίες και η προσθήκη μερικών σύντομων μηνυμάτων θα καλλιτέρευε πολύ τα πράγματα. Ένα τέτοιο που είναι συνήθως γνωστό σαν μήνυμα "prompt", μπορεί εύκολα να προστεθεί χρησιμοποιώντας μια εντολή PRINT. Αυτό έγινε στη σειρά 10 του προγράμματος 1.3. Πληκτρολογήστε τη νέα σειρά και δώστε την εντολή RUN για να τρέξει το πρόγραμμα.

ΠΡΟΓΡΑΜΜΑ 1.3

```
10 PRINT "PLEASE TYPE IN YOUR NAME"
20 INPUT NAME$
30 PRINT NAME$
```

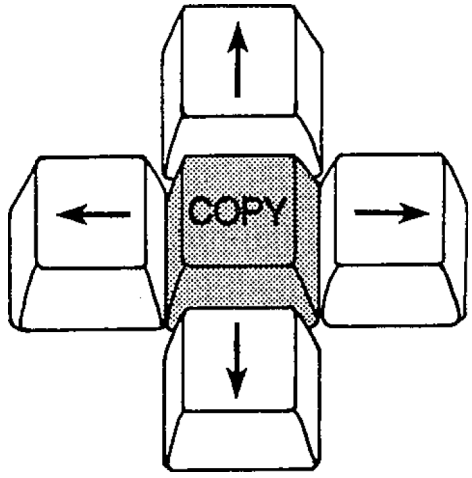
Όταν τρέχει το πρόγραμμα 1.3, σας δίνει κάπως περισσότερες πληροφορίες και ζητείται άμεσα το όνομά σας. Παρ' όλα αυτά υπάρχει ένας πιο κομψός τρόπος για να γίνει αυτό στη BASIC! Η εντολή INPUT μπορεί να χρησιμοποιηθεί μόνη της για να εκτυπώσει ένα μήνυμα. Παρεμβάλουμε το μήνυμα ανάμεσα στο INPUT και το όνομα της μεταβλητής. Η εντολή που δημιουργείται είναι ένα μίγμα των εντολών 10 και 20.

Αυτό που θα κάνουμε είναι να προσθέσουμε την πρόταση σε εισαγωγικά από την εντολή 10 στην εντολή INPUT στην 20. Ο υπολογιστής Amstrad είναι εφοδιασμένος με διορθωτικές δυνατότητες για να σας βοηθήσει να κάνετε τη διόρθωση.

1.1.10.1 EDIT

Για να διορθώσετε τη σειρά 20 πληκτρολογήστε απλά EDIT 20 και πατήστε το ENTER. Θα εμφανιστεί στην οθόνη ένα αντίγραφο της σειράς 20 με το δρομέα τοποθετημένο στο 2 του 20. Τώρα βρίσκεστε στη κατάσταση EDIT.

Πάνω από το τμήμα των αριθμών στο πληκτρολόγιο θα δείτε τα παρακάτω πλήκτρα.



Εικόνα 1.13 Τα πλήκτρα του δρομέα.

Αυτή η ομάδα πλήκτρων μας βοηθάει να αλλάζουμε τις σειρές χωρίς να χρειάζεται να τις ξαναγράψουμε. Τα πλήκτρα αυτά μετακινούν το δρομέα όταν βρισκόμαστε στη κατάσταση EDIT. Χρησιμοποιώντας το πλήκτρο με το δεξί βέλος μετακινείται ο δρομέας μέχρι να πάρει θέση ανάμεσα στην εντολή INPUT και το όνομα της μεταβλητής.

```
20 INPUT NAME$
```

Όταν ο δρομέας βρίσκεται στη σωστή θέση αρχίστε να πληκτρολογείτε τα παρακάτω:

```
"PLEASE TYPE IN YOUR NAME";
```

Καθώς πληκτρολογείτε αυτά, όλα όσα βρίσκονται δεξιά από το δρομέα μετακινούνται προς τα δεξιά. Ότι πληκτρολογείται παρεμβάλλεται ανάμεσα στο INPUT και το NAME$. Όταν πληκτρολογείτε την άνω τελεία το NAME$ κινείται προς τα μπροστά και έτσι εμφανίζεται το σύμβολο του δολλαρίου σε μία νέα σειρά.

```
20 INPUT"PLEASE TYPE IN YOUR NAME"; NAME
$
```

Μη στενοχωριέστε γι' αυτό. Ο Amstrad καταλαβαινει ακόμη ότι αυτό είναι τμήμα της σειράς του προγράμματος. Δεν θα έπρεπε να πατήσετε το ENTER μέχρι να τελειώσετε την πληκτρολόγηση της σειράς. Ο Amstrad σας δίνει τη δυνατότητα να πληκτρολογείτε μέχρι να γεμήσουν εξίμισυ σειρές της οθόνης, μετά θ' ακούσετε ένα μπιπ που θα σημαίνει ότι δεν μπορείτε να πληκτρολογήσετε περισσότερο.

Αν η σειρά είναι τώρα ίδια όπως πιο πάνω έχετε τελειώσει το διόρθωμα της σειράς 20, έτσι πατώντας το πλήκτρο ENTER θα έχετε δημιουργήσει μία νέα σειρά. Θα έχετε επίσης αφήσει την κατάσταση EDIT.

Αυτή ήταν μία άμεση παρεμβολή. Μια λιγότερο άμεση εργασία θα ήταν το ν' αλλάξουμε κάτι όπως το εξής:

```
100 PRINT "THIS MESSAGE IS WRONG"
```

σε

```
100 PRINT "THAT MESSAGE WAS WRONG"
```

Ζητείται να αντικατασταθεί το THIS με το THAT, τότε ο δρομέας χρειάζεται να μετακινηθεί για να αντικαταστήσει το IS με το WAS. Ολόκληρη η εργασία μπορεί να εκτελεστεί σε απλά βήματα.

* Πληκτρολογήστε: 100 PRINT "THIS MESSAGE IS WRONG"

* Πατήστε το ENTER

* Πληκτρολογήστε: EDIT 100

* Χρησιμοποιήστε το σωστό πλήκτρο του δρομέα για να το φέρετε πάνω στο I του THIS

* Πατήστε το πλήκτρο CLR δύο φορές. Πατώντας το CLR σβήνεται οποιοσδήποτε χαρακτήρας, πάνω στον οποίο βρίσκεται ο δρομέας.

* Πληκτρολογήστε: AT

* Πατήστε το σωστό πλήκτρο του δρομέα εννέα φορές, έτσι ώστε να το φέρετε πάνω στο I του IS.

* Πατήστε το CLR μία φορά.

* Πληκτρολογήστε: WA

* Πατήστε το ENTER

* Τώρα έχετε διορθώσει σωστά τη σειρά 100. Συγχαρητήρια!

1.1.10.2 COPY

'Ενας άλλος τρόπος για να διορθώσετε τη σειρά 20 θα ήταν να χρησιμοποιήσετε το πλήκτρο COPY. Για να δείτε πως θα μπορούσε να γίνει αυτό ακολουθείστε τα παρακάτω στάδια:

* Πληκτρολογήστε: LIST 10

* Πατήστε το ENTER

* Ενώ κρατάτε κάτω το SHIFT χρησιμοποιήστε τα πλήκτρα του δρομέα για να το μετακινήσετε μέχρι να έρθει πάνω στο πρώτο εισαγωγικό στη σειρά 10. Τώρα υπάρχουν δύο δρομείς στην οθόνη. Ο ένας πάνω στο εισαγωγικό είναι ο δρομέας για τη διόρθωση (EDIT).

* Πληκτρολογήστε: 20 INPUT Αυτός θα εμφανιστεί εκεί που είναι ο κανονικός δρομέας.

* Πατήστε το πλήκτρο COPY. Αυτό θα μετακινήσει το δρομέα διόρθωσης αντιγράφοντας ότι βρίσκεται κάτω απ' αυτόν μέχρι και τη θέση του δρομέα κειμένου.

* Κρατήστε πατημένο το πλήκτρο COPY μέχρι η σειρά 20 να γίνει όπως παρακάτω:

  20 INPUT "PLEASE TYPE IN YOUR NAME"

* Πληκτρολογήστε: ;NAME$

* Πατήστε το ENTER

* 'Εχετε διορθώσει τη σειρά 20 και πάλι σωστά. Περισσότερα συγχαρητήρια!

Παρόλο που αυτό φαίνεται σαν μια μακριά και δύσκολη διαδικασία, οι διορθωτικές δυνατότητες είναι αρκετά εύκολες στη χρήση, ιδιαίτερα όταν έχει κανείς εξασκηθεί.

Οπωσδήποτε, επιστρέφοντας πίσω στο πρόγραμμα 1.3, έχοντας τροποποιήσει τη σειρά 20 έτσι ώστε να περιλαμβάνει το μήνυμα, το πρόγραμμα τώρα περιέχει το ίδιο μήνυμα δύο φορές και έτσι η σειρά 10 πρέπει να σβηστεί. Αυτό είναι αρκετά εύκολο να γίνει πληκτρολογώντας τον αριθμό 10 και πατώντας μετά το ENTER.

Αφού σβήσετε η σειρά 10, πληκτρολογήστε LIST και το πρόγραμμα 1.4 θα εμφανιστεί όπως παρακάτω:

ΠΡΟΓΡΑΜΜΑ 1.4

```
20 INPUT "PLEASE TYPE IN YOUR NAME";NAME$
30 PRINT NAME$
100 PRINT "THAT MESSAGE WAS WRONG"
```

Όταν αυτό τρέξει θα εμφανιστεί ένα μήνυμα ρωτώντας το όνομά σας και στη συνέχεια αφού δοθεί το δεδομένο, θα εκτυπωθεί απλά το όνομά σας στην οθόνη. Τώρα θα ήταν καιρός να σβήσετε τη σειρά 100, γι' αυτό κάντε το. Πληκτρολογήστε το 100 και πατήστε ENTER.

Ευτυχώς, η εντολή PRINT μπορεί επίσης να περιέχει μία μεταβλητή, κατά τον ίδιο ακριβώς τρόπο όπως η εντολή INPUT, έτσι προσπαθήστε να τροποποιήσετε το πρόγραμμα 1.4 σύμφωνα με τις οδηγίες που δίνουμε στην άσκηση 1.1

ΑΣΚΗΣΗ 1.1

Διορθώστε τη γραμμή 30 του προγράμματος 1.4 έτσι που το πρόγραμμα να ανακοινώνει:

YOUR NAME IS FRED

(ή οποιοδήποτε τυχαίνει να είναι το όνομά σας). Η απάντηση δίνεται στο κεφάλαιο των λύσεων.

Η εντολή INPUT μπορεί να χρησιμοποιηθεί για να εισαχθούν ταυτόχρονα περισσότερα από ένα ξεχωριστά πράγματα. Για να το κάνετε αυτό χρησιμοποιείται ο διαχωριστής κόμμα (,), π.χ.

```
10 INPUT A$,B$,C$
```

Όταν εκτελεστεί το πρόγραμμα, ο υπολογιστής θα περιμένει μέχρι να δοθούν τρία ξεχωριστά δεδομένα.

Υποθέστε ότι τα δεδομένα ήταν ONE, TWO και THREE. Θα πληκτρολογούσατε κανονικά το ONE μετά θα πατούσατε το ENTER (δίνοντας στη μεταβλητή A$ την τιμή ONE), μετά θα πληκτρολογούσατε το TWO και θα πατούσατε το ENTER και το ίδιο με το THREE. Με μία πολλαπλή εντολή INPUT (όπως η παραπάνω) μετά την εισαγωγή της πρώτης τιμής και το πάτημα του ENTER ο Amstrad θα απαντήσει με το μήνυμα Redo from start. Αυτό γίνεται επειδή ο υπολογιστής περιμένει να δεχθεί τρία ξεχωριστά δεδομένα και έχει δοθεί μόνο ένα.

Ο σωστός τρόπος εισαγωγής των δεδομένων είναι να πληκτρολογήσετε το ONE μετά ένα κόμμα το TWO κόμμα και τελικά το THREE ΧΩΡΙΣ ν' ακολουθείται από κόμμα, αλλά πατώντας το ENTER. Το κόμμα λέει στον υπολογιστή ότι το επόμενο ξεχωριστό δεδομένο, πρέπει να δοθεί στην επόμενη μεταβλητή.

Αν η εντολή INPUT ζητά τρία στοιχεία, όπως παραπάνω στη σειρά 10, και εισάγονται τέσσερα δεδομένα, χωρισμένα το καθένα με κόμμα, ο χρήστης θα πάρει το μήνυμα λάθους ?Redo from start και ο υπολογιστής θα περιμένει να πληκτρολογηθούν πάλι τα τρία ξεχωριστά δεδομένα.

Επειδή το κόμμα χρησιμοποιείται για να χωρίζει αλφαριθμητικές μεταβλητές κατά την εισαγωγή τους, δεν μπορεί να είναι μέρος μιας αλφαριθμητικής μεταβλητής που εισάγεται. Η πολλαπλή εντολή INPUT μπορεί να περιέχει και μηνύματα και αριθμούς. Οι ίδιοι κανόνες εφαρμόζονται τόσο σε αριθμητικές μεταβλητές όσο και σε αλφαριθμητικές.

Τα μηνύματα ή prompts που μπορείτε τώρα να βάλετε στα προγράμματά σας, είναι πολύτιμα στην καθοδήγηση του χρήστη ώστε να εισάγει τα σωστά δεδομένα. Προσπαθήστε να λύσετε την άσκηση 1.2 χρησιμοποιώντας ξεκάθαρα μηνύματα.

ΑΣΚΗΣΗ 1.2

Τροποποιήστε το πρόγραμμα που έγινε στην 'Ασκηση 1.1 έτσι που αυτό να ρωτά το όνομα και την ηλικία κάποιου και έπειτα να αναφέρει σ' αυτούς "YOUR NAME IS ..., YOUR AGE IS ...". Μία δυνατή λύση δίνεται στο κεφάλαιο των λύσεων.

1.1.10.3 LOCATE

Τόσο απλά, όπως η εκτύπωση του ονόματός σας στην οθόνη, μπορείτε να διαλέξετε που ακριβώς πάνω στην οθόνη θέλετε να εκτυπωθεί το όνομά σας. Η οθόνη του Amstrad είναι χωρισμένη σε μη ορατές θέσεις, που ονομάζονται κυψέλες και κάθε χαρακτήρας αντιστοιχεί σε μία τέτοια θέση. Υπάρχουν 40 κυψέλες κατά μήκος και 25 προς τα κάτω, αριθμημένες αυστηρά από το 1 μέχρι το 40 και από το 1 μέχρι το 25 αντίστοιχα. Αυτές οι κυψέλες αναγνωρίζονται αναφέροντας την οριζόντια και κάθετη θέση τους, γνωστές σαν συντεταγμένες. Σ' αυτό το βιβλίο θα ακολουθήσουμε τον καθιερωμένο τρόπο να αναφερόμαστε στην οριζόντια διεύθυνση, ονομάζοντάς την συντεταγμένη X και στην κατακόρυφη ονομάζοντάς την συντεταγμένη Y.

Χρησιμοποιώντας την εντολή LOCATE πρέπει να καθορίσετε την θέση (κυψέλη) από την οποία θέλετε ν' αρχίσει ο υπολογιστής την εκτύπωση. Η θέση θα πρέπει να αναφέρεται με τις συντεταγμένες της X και Y, π.χ.

```
LOCATE 10,2
```

Η εντολή αυτή θα τοποθετήσει το δρομέα εκτύπωσης 10 θέσεις στο μήκος και στη δεύτερη σειρά. Το επόμενο αποτέλεσμα που πρόκειται να εκτυπωθεί θ' αρχίσει από εκεί. Η εντολή LOCATE μπορεί να βοηθήσει να τακτοποιήσετε την εμφάνιση της οθόνης σας.

Πριν χρησιμοποιήσετε το LOCATE, θα ήταν μια καλή ιδέα να απαλλαχθείτε από το πρόγραμμα που βρίσκεται τώρα αποθηκευμένο στη μνήμη, δηλαδή το πρόγραμμα 1.4 (όπως είναι τροποποιημένο στις ασκήσεις). Για να το κάνετε αυτό θα χρησιμοποιείτε την εντολή: NEW.

1.1.10.4 NEW

Πληκτρολογώντας την εντολή NEW και πατώντας το ENTER κάνετε τον Amstrad να σβήσει οτιδήποτε υπάρχει στη μνήμη του. Θα σβήσει το πρόγραμμά σας και το περιεχόμενο κάθε μεταβλητής που έχετε ορίσει ή χρησιμοποιήσατε. Για να το δείτε αυτό πληκτρολογήστε NEW και μετά πατήστε το ENTER, μετά πληκτρολογήστε LIST και πατήστε το ENTER ξανά. Δεν υπάρχει πια κανένα πρόγραμμα. Εισάγοντας μια φορά το NEW το πρόγραμμά σας φεύγει εντελώς, έτσι βεβαιωθήτε αν θέλετε να το κάνετε αυτό προτού δώσετε την εντολή NEW.

Τώρα μπορούμε ν' αρχίσουμε να γράφουμε ένα νέο πρόγραμμα χρησιμοποιώντας το LOCATE.

ΠΡΟΓΡΑΜΜΑ 1.5

```
10 LOCATE 10,2
20 PRINT "HELLO"
```

Το πρόγραμμα 1.5 θα εκτυπώσει το HELLO αρχίζοντας από τη δέκατη θέση στο μήκος (X συντεταγμένη) και στη δεύτερη σειρά προς τα κάτω (Y συντεταγμένη). Αν υπήρχε κάτι στην οθόνη γύρω απ' αυτή την περιοχή τότε δεν θα μπορείτε να δείτε το HELLO πολύ καθαρά. Θα ήταν μια καλή ιδέα να καθαρίσετε την οθόνη προτού τρέξει το πρόγραμμα. Η εντολή CLS το κάνει για σας.

1.1.10.5 CLS

Η εντολή CLS χρησιμοποιείται είτε μέσα από ένα πρόγραμμα είτε σαν άμεση εντολή για να καθαρίσει την οθόνη. Εμείς θα τη χρησιμοποιήσουμε στο πρόγραμμά μας για να βεβαιωθούμε ότι θα έχουμε μια καθαρή οθόνη κάθε φορά που θα αρχίζουμε. Κανονικά μετά το CLS οτιδήποτε είναι να εκτυπωθεί θ' αρχίσει στην κορυφή, δηλαδή στην αριστερή γωνία της οθόνης. Παρ' όλα αυτά, η σειρά 10 μετακινεί το δρομέα εκτύπωσης σε διαφορετικό σημείο αρχής σ' αυτό το παράδειγμα, δέκα χαρακτήρες στο μήκος και δύο γραμμές προς τα κάτω.

Με την πρόσθεση της σειράς 5 στο πρόγραμμα γίνεται:

ΠΡΟΓΡΑΜΜΑ 1.6

```
5 CLS
10 LOCATE 10,2
20 PRINT "HELLO"
```

Γράψτε ένα μικρό πρόγραμμα που θα καθαρίζει την οθόνη και έπειτα να ρωτά το όνομά σας, καθαρίστε την οθόνη πάλι και τυπώστε "HELLO FRED" (ή οποιοδήποτε άλλο) στη μέση της οθόνης, χρησιμοποιώντας την εντολή LOCATE. Μία δυνατή απάντηση δίνεται στο κεφάλαιο των λύσεων.

Τώρα έχουμε φτάσει στο τέλος του πρώτου κεφαλαίου και θα έχετε καταλάβει μερικές από τις βασικές εντολές της BASIC του Amstrad.

ΚΕΦΑΛΑΙΟ 2

ΜΕΡΟΣ ΠΡΩΤΟ

2.1.1 Μάντεψε τον αριθμό

Αυτό το πρώτο μικρό έργο θα γίνει για να αναπτύξουμε ένα παιχνίδι που να μαντεύει αριθμούς και να εξετάζει διάφορες τεχνικές χειρισμού αριθμών. Στο παιχνίδι αυτό ο υπολογιστής θα σκέφτεται έναν αριθμό ανάμεσα στο 1 και το 100 και θα ζητείται από τον παίκτη να μαντέψει ποιός είναι ο αριθμός σε λιγότερες από έξι προσπάθειες. Στον παίκτη λέγεται αν ο αριθμός που υπέθεσε είναι πολύ μεγάλος, πολύ μικρός ή σωστός. Μετά από έξι προσπάθειες, εάν δεν έχει μαντέψει σωστά, ο αριθμός θα εμφανιστεί στην οθόνη. Σ' αυτό το στάδιο, ή όταν έχει μαντευθεί ο σωστός αριθμός, ο παίκτης δέχεται την ερώτηση, εάν θέλει να ξαναπαίξει.

Το πρόγραμμα θα κατασκευαστεί σε μια τμηματική μορφή εισάγοντας διάφορες εντολές όπως και όταν είναι αναγκαίο.

2.1.1.1 RND

Σε ένα παιχνίδι σαν αυτό, η ουσιώδης λειτουργία είναι η παραγωγή ενός τυχαίου (δηλαδή απρόβλεπτου) αριθμού για να μαντέψει ο παίκτης. Ο Amstrad χρησιμοποιεί την εντολή RND για να δημιουργήσει έναν τυχαίο αριθμό.

Δοκιμάστε το εξής:

```
PRINT RND
```

Αυτό θα κάνει τον Amstrad να εκτυπώσει έναν τυχαίο δεκαδικό αριθμό ανάμεσα στο 0 και το 1. Αυτός δεν θα είναι ποτέ 0 ή 1, αλλά κάποιος ενδιάμεσος.

Παρ' όλα αυτά, για το παιχνίδι μας το διάστημα αυτό είναι πολύ μικρό, εμείς χρειαζόμαστε μία περιοχή από το 1 μέχρι 100. Αυτό μπορεί να γίνει με απλό πολλαπλασιασμό του τυχαίου αριθμού με το 100, δηλαδή:

```
PRINT 100*RND
```

Τώρα η περιοχή είναι σωστή, αλλά ο αριθμός που θα δημιουργηθεί έχει ψηφία μετά την υποδιαστολή, ενώ αυτό που χρειαζόμαστε είναι ένας ολόκληρος αριθμός δηλαδή

ακέραιος. Η Amstrad BASIC περιέχει μία εντολή για να σβήνει κανείς τα ψηφία μετα την υποδιαστολή, αφήνοντας μόνο το ακέραιο μέρος του αριθμού.

2.1.1.2 INT

Η εντολή για τη δημιουργία ενός ακέραιου αριθμού από ένα δεκαδικό λέγεται INT (INTeger), σ' αυτήν οι παρενθέσεις περιέχουν τον αριθμό ή την έκφραση που θα γίνει ακέραιος. Ο Amstrad το κάνει αυτό αποκόπτοντας τους δεκαδικούς αριθμούς προς το μικρότερο ακέραιο, π.χ.

```
INT(6.0128)=6
INT(5.9)=5
INT(2.3*4)=9
```

Έτσι η εντολή PRINT INT(RND*100) θα εκτυπώσει έναν τυχαίο αριθμό ανάμεσα στο 0 και το 99. Ο λόγος για τον οποίο παράγονται αριθμοί κάτω από το 100 και όχι το 100 είναι ο τρόπος με τον οποίο η εντολή RND παράγει αριθμούς: ποτέ 100 αλλά μερικές φορές 99.99, που, όταν πάρει την ακέραια τιμή του γίνεται 99. Για να παραχθούν αριθμοί στο διάστημα από το 1 μέχρι και το 100 χρειάζεστε απλά να προσθέσετε 1 στην τυχαία τιμή, π.χ.

```
PRINT INT(RND*100)+1
```

Καθώς θα χρησιμοποιηθεί αυτή η γραμμή σε ένα πρόγραμμα, χρειάζεται ένας αριθμός γραμμής. Επίσης χρειαζόμαστε να αποδώσουμε τον τυχαίο αυτόν αριθμό σε μία μεταβλητή, που θα την ονομάσουμε RANUM, από τις λέξεις (RAndom NUMbers) τυχαίοι αριθμοί.

ΠΡΟΓΡΑΜΜΑ 2.1(a)

```
30 RANUM=INT(RND*100)+1
50 PRINT RANUM
```

2.1.1.3 GOTO

Για να επαναληφθούν ένα πλήθος φορές οι γραμμές 30 και 50 χρειάζεται μια εντολή που θα στείλει το πρόγραμμα ξανά πίσω στην αρχή. Η εντολή που το κάνει είναι η "GOTO". Η εντολή GOTO αλλάζει την κατεύθυνση ενός προγράμματος προς ένα δεδομένο αριθμό γραμμής. Έτσι η GOTO εξηγείται από μόνη της και δεν είναι δύσκολο να

γίνει κατανοητή. Μπορεί να προστεθεί στο *πρόγραμμα* 2.1(a) στη γραμμή 80, για να δώσει το πρόγραμμα 2.1(b). Αν γίνει αυτό τότε λέμε ότι το πρόγραμμα 2.1(b) κάνει ανακύκλωση στη σειρά 30 από τη σειρά 80.

ΠΡΟΓΡΑΜΜΑ 2.1(b)

```
30 RANUM=INT(RND*100)+1
50 PRINT RANUM
80 GOTO 30
```

'Οταν αυτό τρέξει, το πρόγραμμα θα μπει σε μια συνεχή ανακύκλωση που θα εκτυπώνει τυχαίους αριθμούς στην οθόνη.

Για να σταματήσετε αυτή τη διαδικασία, πατήστε το πλήκτρο ESCape δύο φορές. Με την πρώτη φορά σταματάτε το πρόγραμμα και με τη δεύτερη βγαίνεται απ' αυτό.

'Ετσι το πρόγραμμα μπορεί να παράγει τυχαίους αριθμούς, αλλά με έναν αρκετά ανεξέλεγκτο τρόπο. Αυτό που χρειαζόμαστε είναι μια μορφή ενός μηχανισμού αρίθμησης και κάποιον έλεγχο σ' αυτή την αρίθμηση που να λέει, για παράδειγμα, πότε έχουν παραχθεί 100 αριθμοί.

'Ενας μηχανισμός αρίθμησης δίνεται με την εισαγωγή μιας μεταβλητής την οποία ας την ονομάσουμε COUNT. Αυτή ορίζεται στην αρχή του προγράμματος ίση με μηδέν και μετά επαυξάνεται κατά 1 κάθε φορά που εκτυπώνεται μία τυχαία τιμή στην οθόνη. 'Ετσι, αν προσθέσουμε τις γραμμές 10 και 70, όπως παρακάτω στο πρόγραμμα, θα μετρήσουμε πόσες φορές έχουμε εκτελέσει το πρόγραμμα. Η δομή του προγράμματος είναι όπως στο πρόγραμμα 2.1(c):

ΠΡΟΓΡΑΜΜΑ 2.1(c)

Τοποθέτηση του μετρητή στο μηδέν	10 COUNT=0
Παραγωγή ενός τυχαίου αριθμού	30 RANUM=INT(RND*100)+1
Εκτύπωση του τυχαίου αριθμού στην οθόνη	50 PRINT RANUM
Αύξηση του μετρητή	70 COUNT=COUNT+1
Επιστροφή πίσω για έναν άλλο τυχαίο αριθμό	80 GOTO 30

Ωστόσο όλα όσα θα κάνει το πρόγραμμα είναι να *μετρήσει πόσοι τυχαίοι αριθμοί* έχουν εκτυπωθεί. Μέχρι τώρα δεν του έχει δοθεί καμμιά εντολή σχετικά με το τι να κάνει ανάλογα με την τιμή του μετρητή. Σαν πείραμα, *αφήστε το πρόγραμμα να τρέξει για λίγα λεπτά.* 'Οταν

βαρεθείτε πατήστε το πλήκτρο ESCape δύο φορές για να βγείτε από το πρόγραμμα. Μετά, για να ελέγξετε αν το πρόγραμμα της μέτρησης έχει λειτουργήσει, πληκτρολογήστε PRINT COUNT και ο υπολογιστής θα σας απαντήσει λέγοντας πόσους τυχαίους αριθμούς έχει εκτυπώσει.

2.1.1.4 IF...THEN

Καλά μέχρι εδώ μπορούμε να μετρούμε. Η επόμενη εργασία είναι να τροποποιήσουμε το πρόγραμμα έτσι ώστε να μπορεί να κάνει έναν έλεγχο στο στάδιο της εκτύπωσης και να σταματάει όταν έχουν εμφανιστεί αρκετοί αριθμοί. Αυτό γίνεται με μία εντολή ελέγχου ή εντολή υπό συνθήκη, που βρίσκεται στη σειρά 60 του προγράμματος 2.1(d).

ΠΡΟΓΡΑΜΜΑ 2.1(d)

```
60 IF COUNT=99 THEN GOTO 90
90 STOP
```

Η γραμμή 60 ελέγχει την τιμή της μεταβλητής COUNT και αν-και ΜΟΝΟΝ αν-αυτή είναι ίση με 99, το πρόγραμμα πηγαίνει στη γραμμή 90 και σταματά.

2.1.1.5 STOP

Η STOP στη γραμμή 90 λέει στο πρόγραμμα να σταματήσει. Στην οθόνη εμφανίζεται το μήνυμα "Break in 90" δηλαδή Διακοπή στη γραμμή 90. Αυτός είναι ο τρόπος με τον οποίο ο Amstrad μας λέει ότι το πρόγραμμα έχει σταματήσει στη γραμμή 90.

Η εντολή IF...THEN επιτρέπει στο χρήστη να τοποθετήσει μία άλλη εντολή της BASIC μετά από ένα IF...THEN. Αυτή η δεύτερη εντολή της BASIC θα εκτελεστεί μόνο όταν η συνθήκη IF εκπληρώνεται.

'Οταν χρησιμοποιούνται εντολές "υπό συνθήκη", όπως στη γραμμή 60, πρέπει να δοθεί προσοχή στον αριθμόν ως προς τον οποίο γίνεται η σύγκριση. Σ' αυτή την περίπτωση η τιμή που τελειώνει το βρόχο ήταν το 99, διότι η αύξηση έγινε μετά την εντολή IF. Αν η εντολή γι' αυτή την αύξηση είχε δοθεί στη γραμμή 55, τότε η συνθήκη στη γραμμή 60 θα είχε εκπληρωθεί όταν COUNT=100.

Ο συνδυασμός των προγραμμάτων 2.1(c) και 2.1(d) δίνει το πρόγραμμα 2.1(e), που όταν τρέξει θα εκτυπώσει 100 τυχαίους αριθμούς.

ΠΡΟΓΡΑΜΜΑ 2.1(e)

```
10 COUNT=0
30 RANUM=INT(RND*100)+1
50 PRINT RANUM
60 IF COUNT=99 THEN GOTO 90
70 COUNT=COUNT+1
80 GOTO 30
90 STOP
```

Όσο τα προγράμματα γίνονται πιο περίπλοκα, γίνονται δύσκολα για να τα παρακολουθήσει κανείς και χρειάζεται να βρεθεί κάποιο μέσο για να αναπαρασταθεί η ροή ενός προγράμματος, σε μια μορφή που μπορεί να γίνει εύκολα κατανοητή. Ένα τέτοιο μέσο είναι γνωστό σαν:

2.1.1.6 ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ

Ένα διάγραμμα ροής αναλύει το πρόγραμμα σε απλά στοιχεία που στο πιο απλό επίπεδο είναι:

1. ΤΕΛΟΣ προγράμματος.

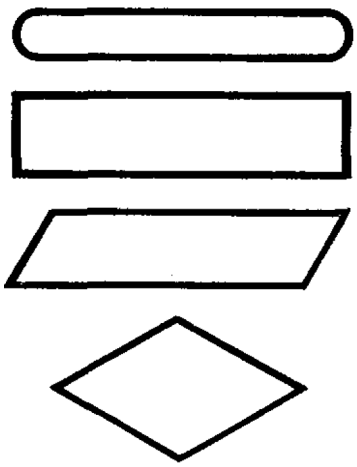
2. Επεξεργασία δεδομένων: LET... εντολές.

3. Είσοδος και Έξοδος: Εντολές όπως PRINT και INPUT.

4. Αποφάσεις: IF...THEN... εντολές.

Υπάρχουν και άλλες εντολές που δεν ταιριάζουν στο παραπάνω υπόδειγμα. Η GOTO, για παράδειγμα, στην πράξη αλλάζει τη σειρά εκτέλεσης των γραμμών ενός προγράμματος ενώ αυτό τρέχει.

Συχνά βοηθάει η χρήση των διαγραμμάτων ροής για να γίνει κατανοητή η λογική ενός προγράμματος. Για κάθε ένα από τα τέσσερα στοιχεία του προγράμματος που αναφέρθηκαν παραπάνω χρησιμοποιούνται καθιερωμένα σύμβολα, καθώς η χρήση τους κάνει την ερμηνεία των διαγραμμάτων ροής πολύ πιο εύκολη.

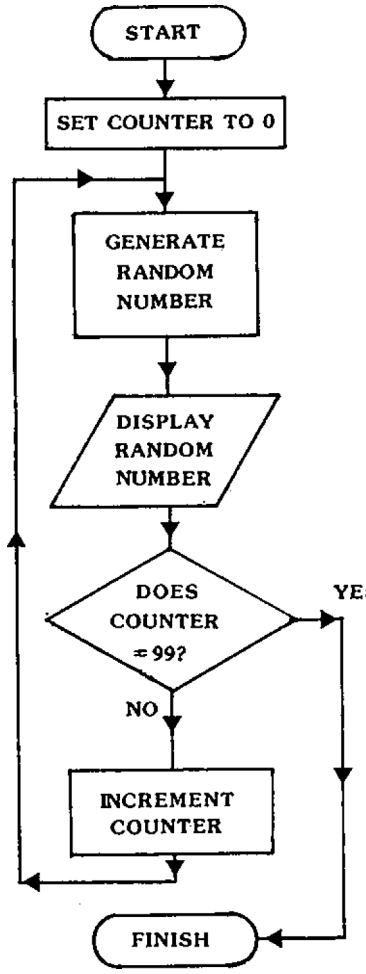


Εικόνα 2.1

Ο κανόνας για να ακολουθήσει κανείς ένα διάγραμμα ροής είναι πραγματικά πολύ απλός. Αρχίζετε από την κορυφή του και ακολουθείτε τις γραμμές που συνδέουν τα σχήματα, μέχρι να φτάσετε στο τέλος. Τα βέλη στις συνδετικές γραμμές δείχνουν τη διεύθυνση της ροής.

Τα διαγράμματα ροής μπορεί να είναι πολύ χρήσιμα όταν πρωτοσχεδιάζεται ένα πρόγραμμα. Όπως συνηθίζεται, οι εξηγήσεις μέσα στα σχήματα πρέπει να γράφονται σε απλή γλώσσα. Είναι κοινό λάθος να γράφεται BASIC στα σχήματα και να νομίζεται ότι το τελικό αποτέλεσμα είναι ένα καθαρό διάγραμμα ροής. Να έχετε πάντα σκοπό να κάνετε τη γλώσσα του διαγράμματος ροής σας και τη γλώσσα του υπολογιστή σας ανεξάρτητες.

Σημειώστε ότι η GOTO στο πρόγραμμα 2.1(e) παριστάνεται με μία γραμμή ροής στο διάγραμμα της εικόνας 2.2 Όλα τα άλλα αντίστοιχα προς τις εντολές του προγράμματος περιλαμβάνονται σε έναν από τους τέσσερεις τύπους σχημάτων που δόθηκαν παραπάνω (Εικόνα 2.1).




```
10    COUNT=0

30    RANUM=INT(RND*100)+1

50    PRINT RANUM

60    IF COUNT=99 THEN GOTO 90

70    COUNT=COUNT+1

90    STOP
```

Εικόνα 2.2 Διάγραμμα ροής του προγράμματος 2.1(e).

Στη BASIC υπάρχουν και άλλοι έλεγχοι συνθηκών και μπορούν να χρησιμοποιηθούν όλα τα γνωστά μαθηματικά σύμβολα πράξεων για να ελεγχθούν διάφορες τιμές. Για παράδειγμα, το πρόγραμμα 2.1(e) μπορεί να τροποποιηθεί για να χρησιμοποιηθεί το σύμβολο μεγαλύτερο από >. Σ' αυτή την περίπτωση σβήστε τη γραμμή 60 και προσθέστε τη γραμμή 75.

ΠΡΟΓΡΑΜΜΑ 2.1(f)

```
75 IF COUNT>99 THEN GOTO 90
```

Ένα άλλο διαθέσιμο σύμβολο για μαθηματικές συγκρίσεις είναι το < που σημαίνει μικρότερο από και χρησιμοποιείται με παρόμοιο τρόπο με το μεγαλύτερο από >.

ΑΣΚΗΣΗ 2.1

Ξαναγράψτε το πρόγραμμα 2.1(e) για να δημιουργήσετε το πρόγραμμα 2.2, το οποίο χρησιμοποιεί τη γραμμή:

75 IF COUNT <(ένας αριθμός) THEN ...

Το πρόγραμμα ακόμη θα τυπώσει σαν έξοδο 100 αριθμούς.

Ζωγραφίστε ένα διάγραμμα ροής για να εξηγήσετε τη λειτουργία του προγράμματός σας. Μία δυνατή απάντηση δίνεται στο κεφάλαιο των λύσεων.

Χρησιμοποιώντας συνθήκες στα προγράμματα είναι δυνατό να δημιουργήσουμε βρόχους, αλλά η BASIC έχει μια εντολή παραγωγής βρόχων που κάνει τη ζωή πολύ πιο εύκολη, αυτή είναι:

2.1.1.7 FOR...NEXT Βρόχος

Όταν χρησιμοποιείτε αυτή η δομή είναι απαραίτητο να ορίσετε την αρχή και το τέλος του βρόχου, όπως φαίνεται παρακάτω:

```
FOR ......   Αρχή του βρόχου
Εντολές μέσα στο βρόχο
NEXT......   Τέλος του βρόχου
```

Όπως στο πρόγραμμα 2.1(e), πρέπει να οριστεί ο αριθμός των ανακυκλώσεων και αυτός έχει οριστεί μέσω μιας μεταβλητής που αυξάνεται αυτόματα με κάθε πέρασμα από το βρόχο. Έτσι η δομή που δείχτηκε πιο πάνω χρειάζεται τροποποίηση, για να γίνει ως εξής:

```
FOR COUNT=1 TO 100
βρόχος
NEXT COUNT
```

Σ' αυτή την εντολή η μεταβλητή για τη μέτρηση, σ' αυτή την περίπτωση η COUNT, είναι γνωστή σαν μεταβλητή του βρόχου ή σαν μεταβλητή ελέγχου επειδή ελέγχει τον αριθμό των περασμάτων από το βρόχο.

Ενσωματώστε την στο πρόγραμμα 2.2 (που δημιουργήθηκε από την άσκηση 2.1). Οι γραμμές FOR και NEXT αντικαθιστούν τις γραμμές 10, 70 και 75 όπως φαίνεται παρακάτω:

ΠΡΟΓΡΑΜΜΑ 2.2(a)

```
10 COUNT=0 <-------------------- FOR COUNT=1 TO 100
30 RANUM=INT(RND*100)+1
50 PRINT RANUM
70 COUNT=COUNT+1 <-------------
75 IF COUNT<100 THEN GOTO 30 <-- NEXT COUNT
```

Σημειώστε τώρα ότι η εντολή STOP δεν είναι αναγκαία καθώς το πρόγραμμα θα σταματήσει όταν ολοκληρωθούν 100 βρόχοι.

Η χρήση της εντολής FOR...NEXT για βρόχους κάνει γενικά ένα πρόγραμμα πιο εύκολα κατανοητό. Για παράδειγμα το πρόγραμμα 2.2(a) μπορεί να απλοποιηθεί όπως φαίνεται στο πρόγραμμα 2.2(b):

ΠΡΟΓΡΑΜΜΑ 2.2(b)

```
10 FOR COUNT=1 TO 100
30 RANUM=INT(RND*100)+1
50 PRINT RANUM
60 NEXT COUNT
```

Ο βρόχος FOR...NEXT πιο πάνω μετράει σε βήματα του 1. Μπορούμε ωστόσο να πούμε στον υπολογιστή να μετράει σε βήματα διαφορετικά από το ένα, χρησιμοποιώντας την εντολή STEP. Αυτή προστίθεται στο τέλος της εντολής FOR όπως:

```
FOR X=1 TO 100 STEP N
```

όπου N το μέγεθος του βήματος.

Αν δεν καθορίσουμε ένα ειδικό μέγεθος βήματος τότε λαμβάνεται σαν δεδομένο, βήμα μεγέθους ένα. Για να δειχτεί η χρήση της εντολής FOR...NEXT...STEP, δώστε και τρέξτε το πρόγραμμα 2.3(a). Πληκτρολογήστε NEW πρώτα για να απαλλαγείτε από το παλιό πρόγραμμα.

ΠΡΟΓΡΑΜΜΑ 2.3(a)

```
10 FOR X=1 TO 100 STEP 2
20 PRINT X
30 NEXT X
```

Ο ιδιαίτερος αυτός βρόχος αρχίζει από το 1 και εκτυπώνει κάθε δεύτερο αριθμό. Έτσι η εκτύπωση θα είναι 1, 3, 5 μέχρι την τελευταία τιμή του X, το 99.

ΑΣΚΗΣΗ 2.2

Αλλάξτε τη γραμμή 10 του προγράμματος 2.3(a) έτσι που ο βρόχος να αρχίζει στο 0 και να αυξάνει με βήματα του τρία. Μία δυνατή απάντηση υπάρχει στο κεφάλαιο των λύσεων.

Ο βρόχος δεν είναι απαραίτητο ν' αρχίσει από το 0 ή το 1. Μπορεί ν' αρχίζει από οποιοδήποτε τιμή μικρότερη από (ή ίση με) την τιμή μετά το TO. Αν η πρώτη τιμή είναι μεγαλύτερη από τη δεύτερη (δηλαδή FOR X=100 TO 50) τότε χρειάζεται μία αντίστροφη μέτρηση: 100, 99, 98, 97 κ.λ.π. Για να το κάνουμε αυτό χρειαζόμαστε μια τιμή βήματος ίση με -1.

Η εντολή για την αντίστροφη μέτρηση πρέπει να διαβάστεί έτσι:

ΠΡΟΓΡΑΜΜΑ 2.3(b)

```
10 FOR X=100 TO 50 STEP -1
```

> ΑΣΚΗΣΗ 2.3
>
> Γράψτε ένα μικρό πρόγραμμα που θα κάνει αντίστροφη μέτρηση με αρχή το "10" με βήματα του "-1". Όταν ο βρόχος θα έχει ολοκληρωθεί τότε το πρόγραμμα θα ΤΥΠΩΣΕΙ "FIRE!". Μία δυνατή απάντηση υπάρχει στο κεφάλαιο των λύσεων.

Αν κάνετε ένα λάθος ενώ χρησιμοποιείτε βρόχους FOR...NEXT και πληκτρολογήσετε κάποια από τις παρακάτω γραμμές, ο υπολογιστής που είναι αρκετά ευαίσθητος θα το αγνοούσε (στην τρίτη περίπτωση θα διέτρεχε το βρόχο μόνο μία φορά).

```
FOR X=100 TO 10 STEP 1
FOR X=10 TO 100 STEP-1
FOR X=10 TO 20 STEP 30
```

Καθώς τα προγράμματα γίνονται πιο πολύπλοκα και περιέχουν τέτοια χαρακτηριστικά όπως ανακυκλώσεις FOR...NEXT, ο κίνδυνος να κάνετε λάθη μεγαλώνει. Ευτυχώς ο Amstrad είναι μαζί σας, όταν σας ξεφύγει κάποιο λάθος, σας το λέει δίνοντάς σας μηνύματα λάθους. Για να το δείτε αυτό προσθέστε τη γραμμή 30 του προγράμματος 2.3(c) στο πρόγραμμα 2.3(b).

ΠΡΟΓΡΑΜΜΑ 2.3(c)

```
30 NEXT K
```

'Οταν εκτελεστεί αυτό ο Amstrad θα δώσει ένα μήνυμα λάθους:

```
Unexpected NEXT in 30
```

Αυτό σας λέει ότι έχετε επιχειρήσει να χρησιμοποιήσετε ένα NEXT χωρίς το αντίστοιχο FOR στη γραμμή 30, καθώς η γραμμή FOR χρησιμοποιεί τη μεταβλητή X και η γραμμή NEXT τη μεταβλητή K.

Λάθη στην Amstrad BASIC προσδιορίζονται με αυτό τον τρόπο καθώς ο υπολογιστής έχει διδαχτεί τη δική του λογική. Για παράδειγμα, αν πείτε στα Αγγλικά "The mat cat, on the sat", αυτό θα' ταν λάθος συντακτικά. 'Ετσι όταν συμβούν παρόμοια λάθη στη γλώσσα του Amstrad, ο υπολογιστής σας λέει ότι υπάρχει "συντακτικό λάθος". Σκεφτείτε ότι τα συντακτικά λάθη είναι ο τρόπος του υπολογιστή να λέει: "I DON'T UNDERSTAND".

'Εχοντας δει μερικές βασικές εντολές BASIC, μπορούμε να ξεκινήσουμε την ανάπτυξη του θέματος αυτού του κεφαλαίου: το παιχνίδι του μαντέματος του αριθμού, που περιγράφτηκε στην αρχή του κεφαλαίου.

Πρώτα πληκτρολογήστε NEW και πατήστε ENTER για να αφαιρέσετε τα προηγούμενα προγράμματα από τη μνήμη του υπολογιστή.

Το πρώτο πράγμα που χρειάζεται το παιχνίδι είναι ένας αριθμός που θα μαντευθεί από μας. Αυτό γίνεται μάλλον πολύ εύκολα θέτοντας μια τυχαία RANUM που, για την ώρα, θα εμφανιστεί στην οθόνη:

ΠΡΟΓΡΑΜΜΑ 2.4(a)

```
30 RANUM=INT(RND*100)+1
35 PRINT RANUM
```

Μετά πρέπει ο παίκτης να εισάγει έναν αριθμό που θα μαντέψει. Ο αριθμός που μάντεψε θα αποθηκευτεί στη μεταβλητή GUESS.

ΠΡΟΓΡΑΜΜΑ 2.4(b)

```
30 RANUM=INT(RND*100)+1
35 PRINT RANUM
50 INPUT GUESS
```

Στο στάδιο αυτό ο αριθμός που μαντέψατε μπορεί να συγκριθεί με τον τυχαίο αριθμό χρησιμοποιώντας την εντολή IF...THEN. Στο προηγούμενο παράδειγμα αυτό χρησιμοποιήθηκε μόνο για να τελειώσουμε το πρόγραμμα μέσω μιας εντολής STOP. Ωστόσο, η εντολή IF...THEN μπορεί ν' ακολουθηθεί από οποιαδήποτε έγκυρη εντολή της BASIC, έτσι στην περίπτωση αυτή η εντολή να μπορεί να λέει: "Αν ο αριθμός που μαντέψατε ισούται με τον τυχαίο αριθμό τότε πες στον παίκτη ότι μάντεψε σωστά". Μεταφράζοντάς το σε BASIC:

```
IF GUESS=RANUM THEN PRINT"WELL DONE-GUESS CORRECT."
```

Μια μικρή συμβουλή πριν προσθέσουμε τη γραμμή αυτή. Κατά τη διάρκεια της ανάπτυξης αυτού του παιχνιδιού πιθανώς θα το τρέξετε εκατοντάδες φορές, αλλά πιθανώς θα γίνει κάπως βαρετό τελικά. Για να το υπερνικήσουμε αυτό, αφήνουμε τη γραμμή 35, δηλαδή την εντολή που τυπώνει την τιμή του τυχαίου αριθμού απείραχτη - μάλιστα το παιχνίδι γίνεται ευκολότερο. Οπότε, μέχρι εδώ το πρόγραμμα έχει διαμορφωθεί ως εξής:

ΠΡΟΓΡΑΜΜΑ 2.4(c)

```
30 RANUM=INT(RND*100)+1
35 PRINT RANUM
50 INPUT GUESS
60 IF GUESS=RANUM THEN PRINT"WELL DONE-G
UESS CORRECT."
```

Στο στάδιο αυτό το πρόγραμμα θα εκτελεστεί και όταν η απάντηση είναι σωστή, δίνει ένα μήνυμα και μετά τελειώνει. Ωστόσο, αν εισαχτεί ένας λανθασμένος αριθμός, το πρόγραμμα θα τελειώσει απλά χωρίς μηνύματα. Για να το χειριστούμε αυτό, προσθέτουμε στις γραμμές 70 και 80 του προγράμματος 2.4(d) δύο ακόμα εντολές υπό συνθήκη.

ΠΡΟΓΡΑΜΜΑ 2.4(d)

```
70 IF GUESS>RANUM THEN PRINT"GUESS TOO L
ARGE-TRY AGAIN."
80 IF GUESS<RANUM THEN PRINT"GUESS TOO S
MALL-TRY AGAIN."
```

'Οταν το πρόγραμμα 2.4(d) εκτελείται, θα χειριστεί μαζί και τις σωστές και τις λανθασμένες απαντήσεις, αλλά μόνο για μια φορά. Με σκοπό να δώσουμε στον παίκτη άλλη μια ευκαιρία, πρέπει να ξαναπάει πίσω στο INPUT (γραμμή 50), αν η απάντηση ήταν λανθασμένη. Αυτή η επιστροφή πρέπει να γίνει υπό συνθήκη βασισμένη στις δοκιμές με IF...THEN που παρουσιάζονται στις γραμμές 60, 70 και 80. Ακόμα μια φορά, η BASIC σώζει την κατάσταση με το ότι μια δεύτερη εντολή της BASIC μπορεί να προστεθεί στο τέλος μιας υπάρχουσας γραμμής αν βέβαια διαχωρίσουμε τα δύο μέρη με άνω και κάτω τελεία. 'Οταν γίνει αυτό η γραμμή αυτή θεωρείται σαν γραμμή πολλαπλών εντολών.

Η δεύτερη εντολή εκτελείται αμέσως, μετά την πρώτη, σαν να ήταν η επόμενη γραμμή, εκτός από το ότι έρχεται μετά από μία εντολή IF...THEN, όπως σ' αυτό το πρόγραμμα. Σ' αυτή την περίπτωση οι εντολές θα εκτελεστούν αν εκτελεστεί η εντολή THEN δηλαδή αν η επιπλέον συνθήκη εκπληρωθεί. 'Ετσι η γραμμή 60 μπορεί να τροποποιηθεί για να γίνει όπως στο πρόγραμμα 2.4(e). Η μεταβολή στη γραμμή 60 μπορεί να γίνει ευκολότερα χρησιμοποιώντας μια επιπλέον ιδιότητα της εντολής EDIT. 'Οπως σε προηγούμενα παραδείγματα, πληκτρολογείτε EDIT και μετά τον αριθμό γραμμής, που στην περίπτωση αυτή είναι 60 και μετά πατήστε ENTER. Μετακινήστε το δρομέα προς το τέλος της γραμμής χρησιμοποιώντας τα πλήκτρα του δρομέα. Μετά πληκτρολογήστε μόνο τις επιπλέον οδηγίες (δηλαδή ":STOP") και πατήστε ENTER. Η νέα γραμμή 60 θα δείχνει κάπως έτσι:

ΠΡΟΓΡΑΜΜΑ 2.4(e)

```
60 IF GUESS=RANUM THEN PRINT"WELL DONE-G
UESS CORRECT.":STOP
```

Αυτή η τροποποίηση θα σταματήσει το πρόγραμμα μετά από μια σωστή απάντηση. Οι γραμμές 70 και 80 μπορούν ομοίως να επεκταθούν στις συγκεκριμένες περιπτώσεις για να ξανακατευθύνουν το πρόγραμμα, όπως στο πρόγραμμα 2.4(f).

ΠΡΟΓΡΑΜΜΑ 2.4(f)

```
70 IF GUESS>RANUM THEN PRINT"GUESS TOO L
ARGE-TRY AGAIN.": GOTO 50
80 IF GUESS<RANUM THEN PRINT"GUESS TOO S
MALL-TRY AGAIN.": GOTO 50
```

Μετά τις τροποποιήσεις στο πρόγραμμα 2.4(f) το παιχνίδι θα επιστρέφει πολλές λανθασμένες προσπάθειες να μαντέψουμε τον αριθμό, αλλά φτάνει σ' ένα STOP όταν δοθεί η σωστή απάντηση.

Αυτό το τέλος είναι μάλλον απότομο και το πρόγραμμα θα μπορούσε να βελτιωθεί σημαντικά αν δινόταν στον παίκτη μια επιλογή μετά από μια σωστή απάντηση να τερματιστεί το παιχνίδι ή να παίξει ξανά. Έτσι μια παραπάνω ρουτίνα προστέθηκε στο τέλος του τρέχοντος προγράμματος δίνοντας έτσι την ευκαιρία στο χρήστη να συνεχίσει. Αυτή η ρουτίνα χρησιμοποιεί την εντολή INPUT μ' ένα μήνυμα και μία συνθήκη – δες πρόγραμμα 2.4(g). Επιπρόσθετα, η STOP θα χρειαστεί να φύγει από τη γραμμή 60 και το πρόγραμμα θα ξαναγυρίσει στην INPUT της γραμμής 110.

ΠΡΟΓΡΑΜΜΑ 2.4(g)

```
60 IF GUESS=RANUM THEN PRINT"WELL DONE-G
UESS CORRECT":GOTO 110
110 INPUT "DO YOU WANT ANOTHER GO(Y/N)";
A$
120 IF A$="Y" THEN XXX
```

Στη γραμμή 110, η INPUT περιμένει μια απάντηση του τύπου YES/NO και τα "(Y/N)" (που βρίσκονται σε παρένθεση) είναι πρόσθετες ενδείξεις που δείχνουν στον παίκτη καθαρά τι δεδομένα περιμένει ο υπολογιστής. Η χρήση τέτοιων παροτρύνσεων κάνει δυνατή τη δοκιμή με μια απλή εντολή INPUT. Στη γραμμή 120, είναι αναγκαίο να ελεγχθεί μόνο το "Y" – που σημαίνει "Yes" γιατί προσδοκάται προφανώς αυτή η επιλογή. Αν η επιλογή δεν είναι "Yes", τότε το πρόγραμμα συνεχίζει για να εκτελέσει την επόμενη γραμμή ή, αν δεν υπάρχει καμμία, να τελειώσει την εκτέλεση. Πρόσεξε τα XXX στη γραμμή 120 αναφέρονται στον αριθμό της γραμμής χωρίς μια "GOTO". Αυτό γίνεται γιατί στη BASIC του Amstrad όταν χρησιμοποιείται μια εντολή IF...THEN ή GOTO εννοείται, αν ακολουθεί την THEN ένας αριθμός εντολής πράγματι

μπορείτε, στην περίπτωσή της GOTO, να κάνετε την εντολή IF...GOTO αντί της IF...THEN, αν αυτό σας φαίνεται πιο προφανές.

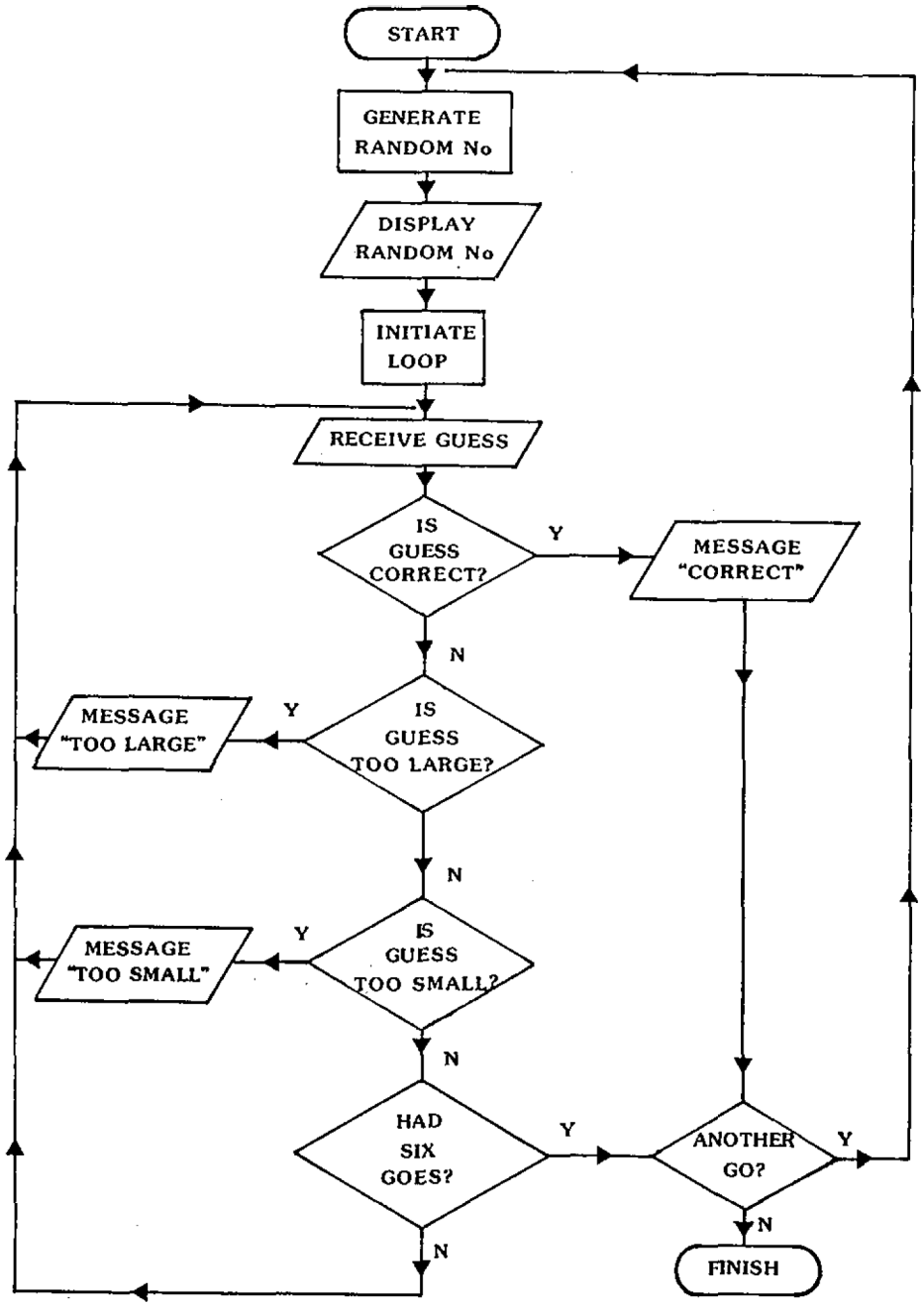
Καθώς το δεδομένο που περιμένουμε να εισαχθεί είναι μία αλφαριθμητική μεταβλητή δηλαδή γράμματα, ήταν αναγκαίο να ορίσουμε ένα κατάλληλο όνομα αλφαριθμητικής μεταβλητής - στην περίπτωση αυτή χρησιμοποιήθηκε η A$. Η γραμμή 120 δεν είναι ολοκληρωμένη και αφήνεται σε σας να τελειώσετε την ανακύκλωση. Η περίπτωση όμως που θα κάνατε κάποιο λάθος, ή δεν είστε σίγουροι, ολοκληρώνεται στις παρακάτω εκδόσεις του παιχνιδιού.

Καθώς είναι γραμμένο το παιχνίδι προς το παρόν, ο παίκτης μπορεί να κάνει όσες προσπάθειες θέλει για να μαντέψει τον αριθμό. Για να προσθέσουμε όμως λίγο παραπάνω ενδιαφέρον, ο αριθμός των προσπαθειών θα περιοριστεί στις έξι. Έχουν ήδη βρεθεί τρόποι για να ανακυκλώνετε το πρόγραμμα ένα δεδομένο αριθμό φορών και όπως στο πρόγραμμα 2.2(b) μπορεί να χρησιμοποιηθεί ένας βρόχος. Αυτό θα υποχρεώσει να επαναληφθεί το μέρος του προγράμματος, που μας ρωτάει ποιον αριθμό μαντέψαμε και που ξεκινάει μετά την παραγωγή του τυχαίου αριθμού στη γραμμή 40. Το κλείσιμο του βρόχου - το "NEXT" - θα λειτουργήσει μετά τις προσπάθειές μας να μαντέψουμε τον αριθμό και πριν από την ερώτηση που γίνεται στη γραμμή 90 "ΑΛΛΗ ΦΟΡΑ;". Αυτά φαίνονται στο πρόγραμμα 2.4(h) όπου η μεταβλητή "COUNT" χρησιμοποιείται για το βρόχο.

ΠΡΟΓΡΑΜΜΑ 2.4(h)

```
30 RANUM=INT(RND*100)+1
35 PRINT RANUM
40 FOR COUNT=1 TO 6
50 INPUT GUESS
60 IF GUESS=RANUM THEN PRINT"WELL DONE-G
UESS CORRECT.":GOTO 110
70 IF GUESS>RANUM THEN PRINT"GUESS TOO L
ARGE - TRY AGAIN.": GOTO 50
80 IF GUESS<RANUM THEN PRINT"GUESS TOO S
MALL - TRY AGAIN.": GOTO 50
90 NEXT COUNT
110 INPUT"DO YOU WANT ANOTHER GO (Y/N)";
A$
120 IF A$="Y" THEN 30
```

Για να καταλάβετε το πρόγραμμα, τρέξτε το για λίγη ώρα, πρώτα με σωστές απαντήσεις και μετά με λανθασμένες για να ελέγξετε το βρόχο. Αν μετρήσετε τις λανθασμένες απαντήσεις θα βρείτε ότι, ο βρόχος δεν είναι ενεργοποιημένος. Για να σας βοηθήσουμε να καταλάβετε τι συμβαίνει σας δίνουμε το διάγραμμα ροής του προγράμματος στο σχήμα 2.3. Μπορείτε να το χρησιμοποιήσετε για να διορθώσετε το 2.4(h). Μην ανησυχήσετε αν δεν μπορείτε, η διόρθωση εξηγείται παρακάτω. Παρεμπιπτόντως σας λέμε ότι υπάρχουν περισσότερα από 4 λάθη, ή "παράσιτα" όπως λέγονται μέσα στο πρόγραμμα αυτή τη στιγμή. Προσπαθείστε να τα βρείτε.


START
GENERATE RANDOM No
DISPLAY RANDOM No
INITIATE LOOP
RECEIVE GUESS
IS GUESS CORRECT?
Y
MESSAGE "CORRECT"
N
MESSAGE "TOO LARGE"
Y
IS GUESS TOO LARGE?
N
MESSAGE "TOO SMALL"
Y
IS GUESS TOO SMALL?
N
HAD SIX GOES?
Y
ANOTHER GO?
Y
N
N
FINISH

Εικόνα 2.3

2.1.2 Επεξήγηση – Μη διαβάσεις αυτό μέχρι να κάνετε μια προσπάθεια μόνοι σας

Αν παρακολουθήσουμε το πρόγραμμα δίνοντας μια σωστή απάντηση φαίνεται ότι δε θα έχουμε φανερά προβλήματα. Όμως στη γραμμή 60 η σωστή απάντηση οδηγεί το πρόγραμμα στη γραμμή 110. Ενώ αυτό είναι λογικά σωστό, έχει σαν αποτέλεσμα να βγούμε έξω από το βρόχο FOR...NEXT, το οποίο είναι κακή πρακτική προγραμματισμού. Η FOR περιμένει για ένα NEXT το οποίο ουδέποτε θα έλθει. Είναι το ίδιο σαν να έχεις κανονίσει να ταξιδέψεις με ένα τρένο και φεύγεις με λεωφορείο!! Μερικοί υπολογιστές θα καταρρεύσουν αν κάνεις κάτι τέτοιο αλλά ευτυχώς ο Amstrad είναι αρκετά έξυπνος και έτσι το πρόγραμμα θα τρέξει κανονικά.

Παρ' όλα αυτά θα τροποποιήσουμε τη γραμμή 60, έτσι ώστε αντί να δώσουμε στη COUNT κάποια τιμή, έστω 99, η οποία θα σημαίνει ότι, όταν η NEXT συμπληρώνεται στη γραμμή 90 ο υπολογιστής θα νομίζει ότι ο βρόχος θα έχει πλήρως συμπληρωθεί. Ακόμη ο βρόχος δεν ενεργοποιείται μετά τις επιτρεπόμενες 6 προσπάθειες. Αν ένα YES είναι σαν απάντηση στην ερώτηση IS GUESS TOO HIGH? τότε έχουμε αποτέλεσμα ένα μήνυμα. Ωστόσο, το πρόγραμμα επιστρέφει από αυτό το σημείο για να επιτρέψει τη νέα εκτέλεση με άλλο INPUT. Η απομάκρυνση του "GOTO 50" στις γραμμές 70 και 80 θα επιτρέψει αυτές τις γραμμές να ακολουθηθούν από τη γραμμή 90, όπου το "επόμενο COUNT" συναντιέται. Η κλήση της "NEXT" ερμηνεύει την απαραίτητη προσαύξηση της "COUNT" και επίσης ελέγχει αν έχει φτάσει ακόμα το 6. Μόλις η ανακύκλωση ολοκληρωθεί, η εντολή NEXT επιτρέπει το πρόγραμμα να τρέξει διαμέσου της γραμμής 110. Στο σημείο αυτό ο παίκτης θα ρωτηθεί: "Θέλεις άλλη φορά;". Πριν από αυτό θα ειπωθεί στον παίκτη "ΕΧΕΙΣ ΤΕΛΕΙΩΣΕΙ ΤΙΣ ΕΞΗ ΦΟΡΕΣ ΣΟΥ" αλλά μόνο αν όλες οι μαντείες ήταν λάθος. Για να πραγματοποιήσουμε μια γραμμή εκτύπωσης (PRINT) θα μπορούσε να προστεθεί στη γραμμή 100 αν η COUNT είναι μικρότερη από 99, που θα υπάρχει αν καμμία από τις μαντείες δεν ήταν σωστή.

Για να συνοψίσουμε, οι 4 τροποποιήσεις που χρειάστηκαν στο πρόγραμμα 2.4(h) είναι:

i. Σβύσε την GOTO 110 στη γραμμή 60 και γράψε COUNT=99

ii. Σβύσε την GOTO 50 στη γραμμή 70

iii. Σβύσε την GOTO 50 στη γραμμή 80

iv. Δώσε τη γραμμή 100.

```
100 IF COUNT<99 THEN PRINT"SORRY, YOU'VE HAD YOUR SIX
GOES."
```

Αυτό δίνει το πρόγραμμα 2.4(i)

ΠΡΟΓΡΑΜΜΑ 2.4(i)

```
30 RANUM=INT(RND*100)+1
35 PRINT RANUM
40 FOR COUNT=1 TO 6
50 INPUT GUESS
60 IF GUESS=RANUM THEN PRINT"WELL DONE -
 GUESS CORRECT.":COUNT=99
70 IF GUESS>RANUM THEN PRINT"GUESS TOO L
ARGE - TRY AGAIN."
80 IF GUESS<RANUM THEN PRINT"GUESS TOO S
MALL - TRY AGAIN."
90 NEXT COUNT
100 IF COUNT<99 THEN PRINT"SORRY, YOU'VE
 HAD YOUR SIX GOES."
110 INPUT"DO YOU WANT ANOTHER GO (Y/N)";
A$
120 IF A$="Y" THEN 30
```

2.1.2.1 IF...THEN...ELSE

Η εντολή IF...THEN χρησιμοποιήθηκε ως τώρα για να δουλεύει μόνο αν μία συνθήκη είναι αληθινή, δηλαδή αν A=5 THEN PRINT "πέντε". Ωστόσο, η IF...THEN έχει μια παραλλαγή που επιτρέπει το πρόγραμμα να προσπεράσει τη γραμμή IF...THEN μόνο στη περίπτωση που η συνθήκη είναι λάθος. Για παράδειγμα, η γραμμή 120 του προγράμματος έχει την ακόλουθη μορφή.

```
IF A$="Y" THEN 30
```

Αυτή ελέγχει μία είσοδο, αν δηλαδή το πλήκτρο που πατήθηκε ήταν "Y" τότε θα ξαναπαίξει οπότε το πρόγραμμα ξαναγυρίζει στη γραμμή 30. Παρ' όλα αυτά, αν το πλήκτρο που πατήθηκε δεν ήταν "Y" τότε το πρόγραμμα απλά τελειώνει. Χρησιμοποιώντας ELSE μπορείς να πεις στον παίκτη ότι το πρόγραμμα τελείωσε στη γραμμή 120 αν οι παίκτες αποφασίσουν ότι δεν θέλουν να ξαναπαίξουν, το πρόγραμμα θα τους πει αντίο και θα τελειώσει. Έτσι η γραμμή 120 πρέπει να περιέχει τα παρακάτω:

```
IF A$="Y" THEN 30:ELSE PRINT"GOODBYE":EN
D
```

Το τμήμα της ELSE θα εκτελεστεί μόνο, αν η A$ δεν είναι "Y". Αν η A$ είναι "Y" τότε το πρόγραμμα επιστρέφει στη γραμμή 30, όπως προηγούμενα.

ΠΡΟΓΡΑΜΜΑ 2.4(j)

```
120 IF A$="Y" THEN 30:ELSE PRINT "GOOD B
YE!":END
```

Εκείνο που απομένει να γίνει τώρα στο πρόγραμμα, είναι να αναφέρει πόσες προσπάθειες έκανε ο παίκτης για να φτάσει στη σωστή απάντηση.

> ΑΣΚΗΣΗ 2.4
>
> Προσθέστε μία συνάρτηση αναφοράς στο πρόγραμμα 2.4(i) έτσι ώστε να λέει στον παίκτη πόσες προσπάθειες έκανε για να πάρει τη σωστή απάντηση. Μία δυνατή απάντηση δίνεται στο επόμενο πρόγραμμα, αλλά δείτε αν μπορείτε να το δουλέψετε μόνοι σας πριν κοιτάξετε.

Μόλις η άσκηση 2.4 ολοκληρωθεί το αποτέλεσμα θα είναι ένας αριθμός λειτουργίας του παιχνιδιού μαντέματος. Από πολλές απόψεις, είναι πολύ απλό, αλλά από το πρόγραμμα 2.5 και πέρα, τα υπόλοιπα εξαρτώνται από αυτό. Η σημαντικότερη βελτίωση που χρειάζεται είναι ένα εισαγωγικό μήνυμα που να λέει στον παίκτη, τι περίπου είναι το παιχνίδι και ποιοί είναι οι κανόνες και ένα ευγενικό goodbye, όταν ο παίκτης τελειώνει. Θα μπορούσε ακόμη να υπάρχει μια ιδέα να σταματά να τυπώνει τον τυχαίο αριθμό στο ξεκίνημα του παιχνιδιού!

Στο γράψιμο του προγράμματος, έγινε μια πρόβλεψη για προσθήκες αργότερα στις γραμμές 10 και 20. Στο πρόγραμμα 2.5 παρακάτω, οι δύο αυτές γραμμές ξεκινούν με

μια εντολή REM που αναγνωρίζει κάθε γραμμή, σαν γραμμή επεξήγησης. Μόλις, ο Amstrad ανακαλύψει μια REM, αγνοεί οτιδήποτε ακολουθεί σ' αυτήν την γραμμή. Μέσω των REMs μπορούν να εισαχθούν σχόλια στα προγράμματα που καθιστούν ικανό, είτε το συγγραφέα προγραμμάτων, είτε το χρήστη να ακολουθήσει την λογική τους πιο εύκολα. Λίγα γενναιόδωρα REMs συστήνονται σ' όλους.

ΠΡΟΓΡΑΜΜΑ 2.5

```
10 REM **NUMBER GUESSER**
20 REM *******GAME*******
30 LET RANUM=INT(RND*100)+1
40 FOR COUNT=1 TO 6
50 INPUT GUESS
60 IF GUESS=RANUM THEN PRINT "WELL DONE
- GUESS CORRECT.":PRINT"YOU TOOK ";COUNT
;" GOES":COUNT=99
70 IF GUESS>RANUM THEN PRINT"GUESS TOO L
ARGE - TRY AGAIN."
80 IF GUESS<RANUM THEN PRINT"GUESS TOO S
MALL - TRY AGAIN."
90 NEXT COUNT
100 IF COUNT<99 THEN PRINT "SORRY, YOU'V
E HAD
YOUR SIX GOES."
110 INPUT "DO YOU WANT ANOTHER GO (Y/N)"
;A$
120 IF A$="Y" THEN 30:ELSE PRINT"GOOD BY
E":END
```

Υπάρχουν πολλοί τρόποι που μπορεί ν' αναπτυχθεί το πρόγραμμα – για παράδειγμα μια λειτουργία, GETTING WARMER/COLDER θα μπορούσε να εισαχθεί αντί για τα μηνύματα TOO LARGE/TOO SMALL. Άλλη ανάπτυξη θα μπορούσε να βελτιώσει την εκτύπωση και τα μηνύματα, για παράδειγμα, λέγοντας σ' έναν ανεπιτυχή παίκτη ποιος ήταν ο αριθμός.

ΑΣΚΗΣΗ 2.5

Τροποποιήστε τη γραμμή 100 του προγράμματος 2.5 να τυπώνει τον αριθμό εάν δεν έχει ήδη βρεθεί. Μια δυνατή απάντηση δίνεται στο κεφάλαιο των λύσεων.

2.1.3 Αποθηκεύοντας ένα πρόγραμμα

Τώρα να σώσουμε αυτό το πρόγραμμα στην ταινία!

Μόλις ένα πρόγραμμα οποιουδήποτε μήκους αναπτυχθεί, γίνεται αγγαρεία να συνεχίζεις να το πληκτρολογείς στον υπολογιστή. Μπορείς, όπως θα δεις, να το σώσεις πάνω σε μονάδα αποθήκευσης και μετά να το ξαναφορτώσεις στη μνήμη όταν σου χρειαστεί. Διαφορετικά από τους περισσότερους οικιακούς υπολογιστές στην αγορά, ο Amstrad έχει ένα δυναμικό κασετόφωνο. Δεν μπορείς να παίζεις μουσική σ' αυτό επειδή είναι ειδικά φτιαγμένο να δουλεύει με τον Amstrad. Υπάρχουν δύο κύρια πλεονεκτήματα με το να'χεις μια δυναμική μονάδα ηχογράφησης. Το πρώτο είναι ότι δεν είσαι υποχρεωμένος να αγοράσεις μια, και το δεύτερο είναι ότι σώζοντας προγράμματα, είναι εξαιρετικά σίγουρο. Συνεπώς δεν υπάρχει εντολή επαλήθευσης στον Amstrad που να ελέγχει αν το πρόγραμμά σου έχει σωθεί σωστά.

Η αποθήκευση σε κασέτα περιγράφεται σαν "μη-πτητική" καθώς δε χρειάζεται καμμία ενέργεια για να κρατήσει αποθηκευμένο το πρόγραμμα. Η περιοχή της μνήμης στον υπολογιστή που τα προγράμματά σου αποθηκεύονται, περιγράφεται σαν "πτητική" διότι, μόλις σβήστεί το μηχάνημα, όλα τα περιεχόμενα της μνήμης χάνονται. Η Amstrad BASIC περιέχει δύο εντολές για αποθήκευση στην κασέτα. Οι εντολές αυτές διαμορφώνουν ένα μέρος του λειτουργικού συστήματος του υπολογιστή, εκείνα τα δυναμικά προγράμματα που κάνουν όλη τη λειτουργία του.

Οι εντολές της BASIC για αποθήκευση, είναι SAVE και LOAD.

2.1.3.1 SAVE και LOAD

Η εντολή που χρησιμοποιείται για το σώσιμο του προγράμματος στην ταινία είναι η SAVE. Δεν είναι πολύ δύσκολο να χρησιμοποιηθεί αλλά πρέπει να μνημονευτούν μερικά σημεία:

* Πάτα τα κουμπιά PLAY και RECORD πριν δώσεις την εντολή SAVE.

* Θυμήσου, ότι τα ονόματα των προγραμμάτων θα πρέπει να έχουν όχι περισσότερα από 16 χαρακτήρες (γράμματα και αριθμούς).

* Θυμήσου, να μη δοκιμάσεις να καταγράψεις στον κενό οδηγό, στην αρχή της ταινίας.

* Είναι καλή συνήθεια να σώζεις σπουδαία προγράμματα σε δύο διαφορετικές ταινίες. Έτσι αν χάσεις τη μία έχεις ένα δεύτερο αντίγραφο. Αυτό είναι γνωστό σαν αντίγραφο backup.

* Βάζε πάντα ετικέτες στις ταινίες, έτσι ώστε να ξέρεις τι υπάρχει σ' αυτές. Αυτό σώζει πολύ χρόνο όταν ψάχνεις για ένα πρόγραμμα που μπορούσε να είναι σε οποιαδήποτε από έναν αριθμό ταινιών.

Τώρα ας υποθέσουμε ότι έχεις να σώσεις ένα πρόγραμμα σε μια κασέτα, για παράδειγμα το παιχνίδι μαντέματος "GUESSER" που έχεις μόλις γράψει. Για να σώσεις αυτό το πρόγραμμα κάνε τ' ακόλουθα:

* Βάλε μια κενή (άγραφη) κασέτα μέσα στο κασετόφωνο, αφού βεβαιωθείς ότι η ταινία έχει περάσει το τμήμα του κενού οδηγού, έτσι ώστε το πραγματικό καφέ τμήμα αντιγραφής της ταινίας να γίνει εμφανές.

* Πάτα μαζί τα πλήκτρα PLAY και RECORD του κασετόφωνου.

* Αν θέλεις να ονομάσεις το πρόγραμμά σου κάπως αλλιώς από GUESSER, θα πρέπει ν' αντικαταστήσεις το δικό σου όνομα προγράμματος ανάμεσα στα εισαγωγικά παρακάτω - αλλά θυμήσου, τ' όνομα δεν θα πρέπει να ξεπερνά σε μήκος τους 16 χαρακτήρες.

* Πληκτρολόγησε SAVE "GUESSER" πάτα ENTER.

* Ο Amstrad θ' αναφέρει:

Press REC και PLAY then any key:

Αν δεν έχεις κιόλας κάνει έτσι, τότε πάτα το REC και το PLAY στην ταινία.

* Πάτα οποιοδήποτε πλήκτρο εκτός από το κόκκινο και το πράσινο πλήκτρο διακοπής.

* Περίμενε ο Amstrad τώρα ας ελπίσουμε ότι σώζει το πρόγραμμα στην ταινία. Θα δεις το ακόλουθο μήνυμα:

Saving GUESSER block 1

* Όταν τελειώσει, το μήνυμα Ready θα εμφανιστεί κάτω από το μήνυμα της αποθήκευσης.

2.1.3.2 LOAD

Μόλις το πρόγραμμα σωθεί στην ταινία, μπορεί να μεταφερθεί πίσω στον υπολογιστή μέσω της εντολής LOAD. Έτσι, για να ξαναφορτώσει το πρόγραμμα που καλείται "GUESSER" χρησιμοποιείται η εντολή:

LOAD "GUESSER"

Μόλις αυτό εισαχθεί θα σας ζητηθεί:

Press PLAY then any key:

Με το πάτημα του πλήκτρου PLAY και μετά οποιουδήποτε άλλου πλήκτρου, στο πληκτρολόγιο του Amstrad, το πρόγραμμα θ' αρχίσει να φορτώνεται.

Αν όλα πάνε καλά, τότε ο Amstrad θα πει ότι βρέθηκε το πρόγραμμα αναφέροντας:

Loading GUESSER block 1

Ένα μπλοκ μπορεί να θεωρηθεί σαν ένα κομμάτι του προγράμματος. Εξαρτώμενο από το μέγεθος του προγράμματος, ο αριθμός των μπλοκς, μπορεί νά'ναι μικρός αριθμός (στη περίπτωση αυτή 1) ή ένας μεγάλος αριθμός. Όσο μεγαλύτερο είναι το πρόγραμμα, τόσο μεγαλύτερος γίνεται ο αριθμός του μπλοκ. Ο αριθμός του μπλοκ εκφράζει πόσο πολύ μνήμη καλύπτει το πρόγραμμα.

Μόλις το πρόγραμμα φορτωθεί, τότε το μήνυμα READY θα φανεί στην οθόνη κάτω από το μήνυμα του φορτώματος.

Αν ενώ φορτώνεται το πρόγραμμα πάρετε το μήνυμα:

Read error

στην οθόνη σου, μην πανικοβληθείς. Τις περισσότερες φορές, αυτό σημαίνει ότι δεν έχετε γυρίσει την ταινία στο ξεκίνημα του προγράμματος. Αν και μετά απ' αυτό παίρνετε ακόμη λάθος, τότε είστε αρκετά άτυχος για να έχετε ένα από τα υπερβολικά σπάνια λάθη που συμβαίνουν. Αν είναι έτσι, πρέπει να σώσετε το πρόγραμμα ξανά. Ένα πρόγραμμα μεγίστου ενδιαφέροντος που πρέπει να θυμάστε, είναι ότι όταν μια φόρτωση είναι επιτυχής, κάθε άλλο πρόγραμμα στη μνήμη του υπολογιστή θα σβηστεί.

Μια πολύτιμη ιδιότητα της BASIC του υπολογιστή, είναι ότι δεν είναι αναγκαίο να ονομάσετε το πρόγραμμα που φορτώνετε. Αν έχετε ξεχάσει τ' όνομα με το οποίο το σώσατε τότε η εντολή

LOAD""

θα φορτώσει το πρώτο πρόγραμμα που θα βρει στην ταινία.

2.1.3.3 CAT

Η τελευταία των εντολών κασέτας του Amstrad είναι η CAT. Η CAT είναι συντομογραφία της CATalogue και χρησιμοποιείται για να εμφανίσει τα ονόματα όλων των αρχείων που σώθηκαν στην ταινία του κασετοφώνου.

Μετά την πληκτρολόγηση της CAT ο Amstrad θ' απαντήσει με:

Press PLAY then any key:

Μόλις πατηθεί ένα πλήκτρο ο Amstrad θα ξεκινήσει ψάχνοντας στην ταινία. Κάθε φορά που θα συναντιέται ένα πρόγραμμα, θα εμφανίζεται το παρακάτω:

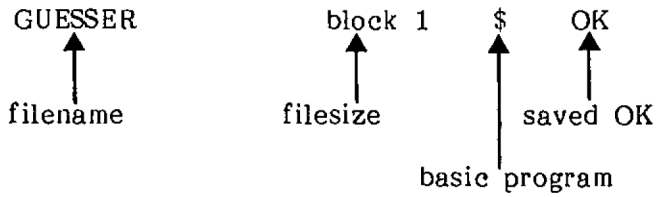
<όνομα αρχείου> μπλοκ <n> <τύπος αρχείου> OK

Όπου <όνομα αρχείου> είναι τ' όνομα του αρχείου που βρέθηκε. <n> είναι ο αριθμός του μπλοκ και <τύπος αρχείου> είναι ένας από τους ακόλουθους χαρακτήρες αρχείων:

$ Πρόγραμμα BASIC
% Προστατευμένο αρχείο BASIC
* ASCII αρχείο κειμένου.
& Δυαδικό αρχείο
' Προστατευμένο δυαδικό αρχείο

Το μήνυμα OK σου λέει ότι το πρόγραμμα σώθηκε ορθά. Το μήνυμα Καταλόγου για την ταινία πάνω στην οποία σώθηκε

το GUESSER θα διάβαζε:



Εικόνα 2.4 Εμφάνιση οθόνης της CAT.

Η Amstrad BASIC περιέχει πολύ λίγες παραλλαγές στις LOAD και SAVE. Όλες αυτές καλύπτονται στο παράρτημα 2.

ΜΕΡΟΣ ΔΕΥΤΕΡΟ

2.2.1 Συγκρίνοντας αριθμούς

Διάφορες τεχνικές είναι επιτρεπτές στη BASIC όταν συγκρίνουμε αριθμούς: μια πολύ χρήσιμη, επιτρέπει να γίνουν δύο συγκρίσεις σε μια εντολή. Χρησιμοποιώντας αυτήν, θ' αναπτυχθεί ένα πρόγραμμα, από το πρόγραμμα 2.5, για να παρουσιάσει ένα παιχνίδι που ρωτά τον παίκτη να μαντέψει δύο αριθμούς. Με σκοπό να το απλοποιήσουμε, ο έλεγχος της ισότητας και οι γραμμές «μεγαλύτερο από» και «λιγότερο από», θα φύγουν, δηλαδή οι γραμμές 60, 70 και 80.

Στη συνέχεια, ένας δεύτερος τυχαίος αριθμός θα πρέπει να εισαχθεί. Θα ονομάσουμε τους δύο αριθμούς RANUM1 και RANUM2, για το RAndom NUMber1 και το RAndom NUMber2. Καθώς ο παίκτης πρόκειται τώρα να ερωτηθεί να μαντέψει δύο αριθμούς θα ήταν επίσης ευκολότερο αν οι πιθανότητες για κάθε αριθμό θα μειώνονταν, στην κλίμακα, ας πούμε 1 έως 4.

ΠΡΟΓΡΑΜΜΑ 2.6(a)

```
30 RANUM1=INT(RND*4)+1:RANUM2=INT(RND*4)
+1
```

Σ' αυτό το συγκεκριμένο παιχνίδι, δύο μαντέματα θα απαιτηθούν και όπως και με τους τυχαίους αυτά μπορούν να ονομαστούν GUESS1 και GUESS2 όπως στη σειρά 50 του προγράμματος 2.6(b).

ΠΡΟΓΡΑΜΜΑ 2.6(b)

```
50 INPUT "GUESS1";GUESS1:INPUT "GUESS2";
GUESS2
```

2.2.1.1 AND

Με δύο μαντέματα και δύο τυχαίους αριθμούς, η διαδικασία ελέγχου γίνεται πολύ περισσότερο πολύπλοκη από αυτή στην αρχή του παιχνιδιού. Ωστόσο για τη BASIC η εντολή AND ευκολύνει τα πράγματα λιγάκι. Δίνει τη δυνατότητα σε κάποιο, για παράδειγμα να συγκρίνει δύο εικασίες πάνω σε μία σειρά ενός προγράμματος. Έτσι, χρησιμοποιώντας την AND είναι πιθανό να πούμε:

```
IF GUESS1=RANUM1 AND GUESS2=RANUM 2 THEN PRINT
"WELL DONE"
```

Το "WELL DONE" θα τυπωθεί μόνο όταν η πρώτη σας μάντεψη είναι ίση με τον πρώτο τυχαίο αριθμό AND και η δεύτερη είναι ίση με το δεύτερο τυχαία παραγμένο αριθμό. Η παραπάνω σειρά ελέγχει για τις σωστές μαντέψεις και μπορεί να χρησιμοποιηθεί μέσα στο πρόγραμμα στη σειρά 60. Αν οι μαντέψεις είναι σωστές τότε ο μετρητής βρόχου τοποθετείται σε μία ακρότατη τιμή και το πρόγραμμα πηδάει στη σειρά 90, όπου το NEXT συναντάται και ο βρόχος τερματίζεται καθώς το COUNT είναι μεγαλύτερο από το 6.

ΠΡΟΓΡΑΜΜΑ 2.6(c)

```
60 IF GUESS1=RANUM1 AND GUESS2=RANUM2 TH
EN PRINT"WELL DONE - GUESS CORRECT":COUN
T=99:GOTO 90
```

Θα ήταν καλό αν ο παίκτης μπορούσε να κερδίσει έχοντας πληκτρολογήσει τους σωστούς αριθμούς, αλλά με λάθος σειρά. Για να γίνει αυτό η γραμμή 65 του προγράμματος συγκρίνει το GUESS1 με το RANUM2 και το GUESS2 με το RANUM1. Αν είναι και τα δύο ίσα τότε ο παίκτης έχει μαντέψει σωστά.

ΠΡΟΓΡΑΜΜΑ 2.6(d)

```
65 IF GUESS1=RANUM2 AND SUESS2=RANUM1 TH
EN PRINT"WELL DONE - GUESS CORRECT":COUN
T=99:GOTO 90
```

Έχοντας εξετάσει για σωστές απαντήσεις είναι καιρός τώρα να ρίξουμε μία ματιά στις πιθανότητες λανθασμένων μαντέψεων. Μιλώντας λογικά αν οι δύο δεν ήταν σωστές, τότε είναι λανθασμένες και πρέπει να ειπωθεί στον παίκτη ότι το έχει κάνει λάθος. Αυτός είναι ο εύκολος δρόμος, ένα μονοπάτι που δε θα πάρουμε. Αντί αυτού, παραπέρα έλεγχοι θα γίνουν πάνω στις μαντέψεις του παίκτη για να δούμε αν μία ή και οι δύο απ' αυτές είναι λάθος.

Ο έλεγχος για δύο λάθος μαντέψεις είναι:

If GUESS1 not equal to RANUM1 AND GUESS1 not equal to RANUM2 AND GUESS2 not equal to RANUM1 AND GUESS2 not equal to RANUM2 then PRINT "Both wrong"

Στη BASIC "όχι ίσο προς" αντιπροσωπεύεται από το μικρότερο από < και μεγαλύτερο από > σύμβολα τοποθετημένα μαζί δηλαδή

<>

Το πρόγραμμα 2.6(e) περιλαμβάνει τη γραμμή η οποία εξετάζει να δει αν οι μαντέψεις είναι λάνθασμένες. Η γραμμή 100 έχει επίσης διορθωθεί για να τυπώνει και τα δύο νούμερα αν αυτά δεν έχουν μαντευθεί μετά από έξι προσπάθειες.

ΠΡΟΓΡΑΜΜΑ 2.6(e)

```
30 RANUM1=INT(RND*4)+1:RANUM2=INT(RND*4)
+1
40 FOR COUNT=1 TO 6
50 INPUT"GUESS1";GUESS1:INPUT"GUESS2";GU
ESS2
60 IF GUESS1=RANUM1 AND GUESS2=RANUM2 TH
EN PRINT"WELL DONE - GUESS CORRECT":COUN
T=99:GOTO 90
65 IF GUESS1=RANUM2 AND GUESS2=RANUM1 TH
EN PRINT"WELL DONE - GUESS CORRECT":COUN
T=99:GOTO 90
70 IF GUESS1<>RANUM1 AND GUESS1<>RANUM2
AND GUESS2<>RANUM1 AND GUESS2<>RANUM2 TH
EN PRINT "BOTH WRONG":GOTO 90
90 NEXT COUNT
100 IF COUNT<99 THEN PRINT "SORRY, YOU'V
E HAD YOUR SIX GOES":PRINT "THE NUMBERS
WERE"; RANUM1,RANUM2
110 INPUT"DO YOU WANT ANOTHER GO (Y/N)";
A$
120 IF A$="Y" THEN 30:ELSE PRINT"GOOD BY
E":END
```

Ως εδώ τα μαντέματα έχουν δοκιμαστεί με σκοπό να δούμε κατά πόσο είναι και οι δύο σωστές ή και οι δύο λάθος. Αν καμμία απ' αυτές τις προϋποθέσεις δεν είναι αληθινή τότε το πρόγραμμα εκτρέπεται στη γραμμή 90 όπου ο μετρητής επαυξάνεται. Τότε μένετε μ' ένα ζευγάρι μαντεμάτων ένα από τα οποία το ένα είναι σωστό και αρκετά λογικό και το άλλο λανθασμένο. Δεν είναι αναγκαίο να κάνετε καμμία παραπέρα δοκιμή διότι τα μαντέματα αν δεν είναι και τα δύο σωστά ή και τα δύο λάθος, τότε ένα απ' αυτά πρέπει να είναι ορθό. Η γραμμή 80 το αναφέρει στον παίκτη.

ΠΡΟΓΡΑΜΜΑ 2.6(f)

```
80 PRINT"ONE RIGHT"
```

Μόλις εισαχθεί η γραμμή 80 έχετε ένα λειτουργικό παιχνίδι μαντέματος δύο αριθμών με σκοπό να το χαρούν και να το απολαύσουν οι φίλοι σας.

2.2.1.2 OR

Όπως περιγράφεται παραπάνω, η εντολή AND μας επιτρέπει να συγκρίνουμε δύο μεταβλητές και να ενεργήσουμε πάνω στ' αποτέλεσμα. Η BASIC μας εξασφαλίζει και μια δεύτερη εντολή, με σκοπό να τη χρησιμοποιήσουμε όταν συγκρίνουμε αριθμούς. Η εντολή OR δουλεύει όπως αυτό:

"Αν αυτό είναι σωστό OR αν το άλλο είναι σωστό τότε κάνε κάτι"

Στη BASIC θα ήταν αυτό:

```
IF A=1 OR A=2 THEN PRINT A
```

Η παραπάνω υπόθεση θα τύπωνε την τιμή του A, αν η τιμή ήταν 1 ή 2. Αυτή είναι μια πιο ευλύγιστη εντολή από την AND.

Η εντολή OR μπορεί να χρησιμοποιηθεί στο παιχνίδι του μαντέματος του αριθμού, με σκοπό να συνδυάσει τις γραμμές 60 και 65, τις δοκιμές των σωστών μαντεμάτων. Αυτό, θα παραγάγει την ακόλουθη γραμμή:

ΠΡΟΓΡΑΜΜΑ 2.7

```
60 IF (GUESS1=RANUM1 AND GUESS2=RANUM2)
OR(GUESS1=RANUM2 AND GUESS2=RANUM1) THEN
 PRINT"WELL DONE GUESS CORRECT":COUNT=99
```

Οι παρενθέσεις στη γραμμη 60 χρησιμοποιούνται για να χωρίσουν κάθε τμήμα των δοκιμών. Οι δοκιμές στις πρώτες παρενθέσεις, είναι εκείνες που ήταν στην αρχική γραμμή 60 και το δεύτερο ζευγάρι των παρενθέσεων περιέχουν τη δοκιμή που ήταν στη γραμμή 65. Η γραμμή 120 δοκιμάζει, για να δει αν ο χρήστης πληκτρολόγησε "Y" για επανάληψη. Ωστόσο, αν εισαχθεί ένα μικρό y δεν θα υπάρξει ταίριασμα και το πρόγραμμα θα τελειώσει. Για να το φροντίσουμε και αυτό το ίδιο καλά, μπορούμε να χρησιμοποιήσουμε ένα "OR" και δοκιμή για y. Αυτό, δίδει την ακόλουθο γραμμή:

ΠΡΟΓΡΑΜΜΑ 2.8

```
120 IF A$="Y" OR A$="y" THEN 30:ELSE PRI
NT" GOODBYE":END
```

Το πρόγραμμα, είναι τώρα ολοκληρωμένο και θα φαίνεται σαν το πρόγραμμα 2.9. Προσπαθείστε να το βελτιώσετε, αυξάνοντας το όριο των αριθμών που θα μαντευτούν, καλύτερες εκτυπώσεις, χρησιμοποιώντας την εντολή LOCATE, βελτιωμένα μηνύματα κ.λ.π.

ΠΡΟΓΡΑΜΜΑ 2.9

```
10 REM NUMBER GUESSER
20 CLS
30 RANUM1=INT(RND*4)+1:RANUM2=INT(RND*4)
+1
40 FOR COUNT=1 TO 6
50 INPUT"GUESS1";GUESS1:INPUT"GUESS2";GU
ESS2
60 IF (GUESS1=RANUM1 AND GUESS2=RANUM2)
OR(GUESS1=RANUM2 AND GUESS2=RANUM1) THEN
PRINT"WELL DONE GUESS CORRECT":COUNT=99:
GOTO 90
70 IF GUESS1<>RANUM1 AND GUESS1<>RANUM2
AND GUESS2<>RANUM1 AND GUESS2<>RANUM2 TH
EN PRINT "BOTH WRONG":GOTO 90
80 PRINT"ONE RIGHT"
90 NEXT COUNT
100 IF COUNT<99 THEN PRINT"SORRY YOU'VE
HAD YOUR SIX GOES.":PRINT"THE NUMBERS WE
RE ";RANUM1;" AND ";RANUM2
110 INPUT"DO YOU WANT ANOTHER GO (Y/N)";
A$
120 IF A$="Y" OR A$="y" THEN 30:ELSE PRI
NT "GOODBYE"END
```

ΜΕΡΟΣ ΤΡΙΤΟ

2.3.1 Μαθηματική προτεραιότητα

Στα Μαθηματικά, υπάρχουν κανόνες που καθορίζουν τη σειρά με την οποία γίνονται οι υπολογισμοί. Οι πολλαπλασιασμοί και οι διαιρέσεις εκτελούνται πάντα πριν τις προσθέσεις και τις αφαιρέσεις δηλαδή έχουν προτεραιότητα. Για παράδειγμα, στους υπολογισμούς:

```
5+3*4
```

ο υπολογισμός 3*4 θα εκτελεστεί πριν προστεθεί το πέντε. Όμοια στους υπολογισμούς:

```
10/2-3
```

Η διαίρεση 10/2 θα υπολογιστεί πρώτα. Αν ένας υπολογισμός έχει πολλαπλασιασμό και διαίρεση σαν αυτόν:

```
3/4*2
```

Τότε το σύνολο υπολογίζεται ξεκινώντας από τ' αριστερά. Ο πρώτος υπολογισμός είναι 3/4 και μετά γίνεται ο πολλαπλασιασμός επί 2 (2*3/4=15).

2.3.2 Παρενθέσεις

Υπάρχει ένας τρόπος να διακόψετε τη μαθηματική προτεραιότητα και αυτός είναι η χρήση των παρενθέσεων. Οι υπολογισμοί που περικλείονται από παρενθέσεις γίνονται πάντα πρώτοι. Για παράδειγμα, στο ακόλουθο σύνολο:

```
2*3+(4-2)
```

4-2=2 θα υπολογιστεί πρώτο. Η επόμενη προτεραιότητα δίνεται στον πολλαπλασιασμό (2*3=6) και τελευταία η πρόσθεση (6+2=8). Στην ύπαρξη των παρενθέσεων, μέσα σε παρενθέσεις γίνονται πρώτα οι υπολογισμοί μέσα στο εσώτατο ζευγάρι των παρενθέσεων και μετά ο υπολογισμός λειτουργεί προς τα έξω. Για παράδειγμα:

```
4*2+(6+4*(3+2)+1)*2
```

Το (3+2) γίνεται δίνοντας 5. Μετά το 4*5, δίνοντας 20. Μετά προστίθεται το 6, μετά το 1, δίνοντας 27. Στο στάδιο αυτό, ο υπολογισμός διαβάζει:

```
4*2+27*2
```

Έτσι ο υπολογισμός 8+54, δίνοντας 62.

Αν ο υπολογισμός περιέχει αρκετά ζευγάρια από παρενθέσεις (όχι αναγκαστικά τη μία μέσα στην άλλη), τότε θα τους κάνει από αριστερά προς τα δεξιά - οπωσδήποτε πρώτα, κάθε ζευγάρι εσωτερικών παρενθέσεων.

Συνοψίζοντας, η μαθηματική σειρά προτεραιότητας είναι:

εσώτατες παρενθέσεις
παρενθέσεις
* και /
+ και −

Σε περίπτωση, που δεν υπάρχει συναγωνισμός για προτεραιότητα (π.χ. υπολογισμός που περιέχει μόνο + και −) τότε υπολογίζεται από αριστερά προς τα δεξιά.

ΚΕΦΑΛΑΙΟ 3

ΜΕΡΟΣ ΠΡΩΤΟ

3.1.1 Γραφικά

Στα δύο προηγούμενα κεφάλαια είδαμε πως εμφανίζονται οι χαρακτήρες πάνω στην οθόνη. Σε αυτό το κεφάλαιο θα σας δείξουμε πως παράγονται εικόνες χρησιμοποιώντας μερικές από τις εντολές του Amstrad για γραφικά υψηλής διακριτικότητας.

3.1.2 Τρόποι

Ο Amstrad έχει τρεις διαφορετικούς τρόπους, δηλαδή μορφές εμφάνισης της οθόνης που ονομάζεται MODE 0, MODE 1 και MODE 2. Όταν ο Amstrad βρίσκεται σε λειτουργία πηγαίνει στο MODE 1. Αυτός είναι ο προκαθορισμένος τρόπος και είναι μάλλον αυτός ο τρόπος που χρησιμοποιείται αυτή τη στιγμή.

Η πιο εμφανής διαφορά ανάμεσα σε κάθε τρόπο είναι το μέγεθος των χαρακτήρων. Το σχήμα 3.1 απεικονίζει τα διαφορετικά μεγέθη των χαρακτήρων για κάθε τρόπο. Το MODE 0 έχει πολύ μεγάλους χαρακτήρες, το MODE 1 μετρίου μεγέθους χαρακτήρες και το MODE 2 μικρού μεγέθους χαρακτήρες.

MODE0 MODE1 MODE2

Εικόνα 3.1 Τα τρία μεγέθη χαρακτήρων.

Για να αλλάξετε κάποιο από τα MODE πληκτρολογείστε απλά MODE και μετά τον επιθυμητό αριθμό του MODE. Για παράδειγμα

```
MODE 0
```

Αυτό θα κάνει τον υπολογιστή να πάει στο MODE 0. Προσέξτε τα μεγάλα γράμματα. Εκτός από το ότι έχει αλλάξει τον τρόπο η οθόνη έχει καθορισθεί και ο δρομέας να είναι τοποθετημένος στην πάνω αριστερή γωνία της οθόνης. Αυτή αναφέρεται σαν η "home" θέση.

Επειδή οι χαρακτήρες του MODE 0 είναι τόσο μεγάλοι μπορούν να μπουν μόνο 20 τέτοιοι χαρακτήρες σε μια σειρά της οθόνης, ακόμα και αν υπάρχουν 25 σειρές. Μια άλλη διαφορά ανάμεσα στους τρόπους είναι το διαφορετικό πλήθος χρωμάτων, που είναι διαθέσιμο κάθε φορά. Το σχήμα 3.2 δείχνει το διαφορετικό πλήθος χρωμάτων και χαρακτήρων για κάθε τρόπο.

MODE	Colours Available	Text Display
0	16	25 lines x 20 characters
1	4	25 lines x 40 characters
2	2	25 lines x 80 characters

Εικόνα 3.2 Η εμφάνιση οθόνης διαθέσιμη σε κάθε MODE.

Ως τώρα χρησιμοποιήσατε τα χρώματα με τα οποία ξεκινά ο Amstrad. Είναι δυνατό για το χρήστη να αλλάξει τα χρώματα της οθόνης και τα χρώματα των χαρακτήρων. Ο Amstrad έχει 27 χρώματα, από τα οποία μπορείτε να διαλέξετε. Έτσι θα μπορέσετε να βρείτε κάποιο που να σας αρέσει. Όσοι από εσάς δεν έχουν έγχρωμη οθόνη θα βλέπουν κάθε διαφορετικό χρώμα σε διαφορετικές αποχρώσεις του πράσινου.

Ο Amstrad είναι πλήρης με τέσσερις εντολές χρωμάτων, οι οποίες είναι οι:BORDER, INK, PAPER και PEN.

3.1.2.1 BORDER

Η εντολή αυτή χρησιμοποιείται για να αλλάξει το χρώμα του πλαισίου. Το χρώμα το οποίο επιθυμείτε δείχνεται με έναν αριθμό που ακολουθεί την εντολή BORDER. Τα 27 χρώματα του Amstrad είναι αριθμητικά κωδικοποιημένα και κυμαίνονται από 0 για το μαύρο ως 26 για το φωτεινό άσπρο. Το χρώμα που χρησιμοποιείται στο πλαίσιο είναι ξεχωριστά από τα διαθέσιμα χρώματα για την εσωτερική περιοχή της οθόνης. Ένας πλήρης κατάλογος των χρωμάτων και των αριθμητικών κωδίκων τους δίνεται στο σχήμα 3.3.

0	BLACK	14	PASTEL BLUE
1	BLUE	15	ORANGE
2	BRIGHT BLUE	16	PINK
3	RED	17	PASTEL MAGENTA
4	MAGENTA	18	BRIGHT GREEN
5	MAUVE	19	SEA GREEN
6	BRIGHT RED	20	BRIGHT CYAN
7	PURPLE	21	LIME GREEN
8	BRIGHT MAGENTA	22	PASTEL GREEN
9	GREEN	23	PASTEL CYAN
10	CYAN	24	BRIGHT YELLOW
11	SKY BLUE	25	PASTEL YELLOW
12	YELLOW	26	BRIGHT WHITE
13	WHITE		

Εικόνα 3.3 Τα χρώματα και οι κώδικες του Amstrad.

Έτσι για να αλλάξετε το πλαίσιο στο ροζ χρώμα πληκτρολογείστε απλά:

```
BORDER 16
```

Με το πάτημα του ENTER το πλαίσιο θα γίνει ροζ. Κατ' αυτόν τον τρόπο το χρώμα του πλαισίου μπορεί να πάρει οποιοδήποτε από τα διαθέσιμα χρώματα.

Ένα ιδιαίτερο χαρακτηριστικό της οθόνης του Amstrad είναι ότι μπορείτε να αλλάξετε το πλαίσιο σε δύο χρώματα. Αυτό γίνεται προσθέτοντας μια άλλη τιμή χρώματος μετά την πρώτη.

```
BORDER 16,0
```

Αυτό προκαλεί το στιγμιαίο αναβόσβημα του Amstrad στο χρώμα του πλαισίου από ροζ σε μαύρο. Ένα πολύ ενδιαφέρον αποτέλεσμα!

3.1.2.2 INK

Η εντολή INK χρησιμοποιείται για να επιλέξετε ποια χρώματα επιθυμείτε να χρησιμοποιήσετε, για την οθόνη και τους χαρακτήρες. Η εντολή αυτή παίρνει τη μορφή:

```
INK i,c
```

όπου "i" είναι ο αριθμός μελάνης (INK) και "c" είναι ο αριθμός χρώματος. Για να καταλάβετε αυτήν την εντολή θα πρέπει να φανταστείτε μια σειρά από μελανοδοχεία. Κάθε μελανοδοχείο έχει το δικό του μοναδικό αριθμό (τον αριθμό μελάνης) και περιέχει το χρώμα "c".

Έτσι το INK 0,6 σημαίνει χαμηλώστε το μελανοδοχείο με τον αριθμό 0 με χρώμα 6 (φωτεινό κόκκινο). Όταν ο Amstrad τοποθετείται σε λειτουργία δίνονται στα μελανοδοχεία προκαθορισμένες τιμές χρωμάτων. Το χρώμα σε κάθε μελανοδοχείο διαφέρει ανάλογα από τη μορφή στην οποία βρίσκεστε. Το σχήμα 3.4 δείχνει τις προκαθορισμένες τιμές μελάνης για κάθε έναν από τους τρεις τρόπους.

INK	MODE 0	MODE 1	MODE 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	Flashing 1,24	20	1
15	Flashing 16,11	6	24

Εικόνα 3.4 Οι τιμές του INK.

Ρίχνοντας μια ματιά στο διάγραμμα θα παρατηρήσετε ότι το MODE 2 έχει μόνο δύο χρώματα, το MODE 1 τέσσερα χρώματα και το MODE 0 έχει 16 χρώματα. Αυτοί είναι οι διαφορετικοί περιορισμοί χρωμάτων για κάθε τρόπο. Δεν έχει σημασία ποια είναι τα συγκεκριμένα χρώματα, αλλά σας επιτρέπονται μόνο, δύο στο MODE 2, τέσσερα στο MODE 1 και δεκαέξι στο MODE 0.

3.1.2.3 PAPER

Η εντολή PAPER αλλάζει το χρώμα του φόντου των χαρακτήρων στην οθόνη και παίρνει τη μορφή:

```
PAPER i
```

όπου "i" είναι το νούμερο της μελάνης (INK). Για παράδειγμα:

```
PAPER 8
```

θα αλλάξει το φόντο, στο χρώμα που τοποθετήθηκε στο μελανοδοχείο 8. Το χρώμα αυτό εξαρτάται από τον τρόπο στον οποίο βρίσκεστε. Τα χρώματα είναι θαλασσί, μπλε και μπλε για τα mode 0, 1, και 2 αντίστοιχα.

Αν τώρα πληκτρολογήσετε μερικούς χαρακτήρες θα δείτε ότι έχουν ένα διαφορετικό χρώμα φόντου από τους άλλους που βρίσκονται στην οθόνη. Για να αλλάξετε το χρώμα του φόντου για ολόκληρη την οθόνη πληκτρολογείστε μόνο "CLS". Ολόκληρη η οθόνη καθαρίζει και κάθε μια κυψέλη χαρακτήρων έχει γεμίσει με καινούριο χρώμα φόντου.

3.1.2.4 PEN

Αυτή αλλάζει το χρώμα του δρομέα και ακολούθως οποιουδήποτε χαρακτήρες που εμφανίζονται ΑΦΟΤΟΥ θα έχει χρησιμοποιηθεί η εντολή PEN. Η PEN δουλεύει με τον ίδιο τρόπο με την PAPER. Έτσι:

```
PEN 3
```

Θα αλλάξει το χρώμα του χαρακτήρα σ' οποιοδήποτε περιέχει το μελανοδοχείο 3.

Τώρα μπορούμε να αλλάξουμε το χρώμα της οθόνης και των χαρακτήρων σ' όποιο συνδυασμό απαιτούμε. Ας υποθέσουμε ότι ζητήσατε μια λαχανί οθόνη με ένα χρώμα μελάνης σε φωτεινό γαλάζιο. Ο Amstrad χρησιμοποιεί το INK 0 σαν την προκαθορισμένη τιμή για το χρώμα της οθόνης και το INK 1 σαν την προκαθορισμένη τιμή για το χρώμα του χαρακτήρα. Έτσι για να αλλάξουμε τα τωρινά χρώματα στις απαιτούμενες τιμές αλλάζουμε απλά την τιμή του INK 0 και του INK 1. Πληκτρολογείστε τις παρακάτω εντολές απευθείας εισόδου:

```
INK 0,21

INK 1,2
```

21 και 2 είναι οι κωδικοί για το λαχανί και το μπλε αντίστοιχα.

ΑΣΚΗΣΗ 3.1

Γράψτε ένα μικρό πρόγραμμα που θα αλλάζει τον υπολογιστή στη MODE 0. Το χρώμα του πλαισίου θα γίνει άσπρο και του φόντου γαλαζοπράσινο. Το χρώμα του χαρακτήρα θα γίνει απαλό βαθυκόκκινο. Μία δυνατή απάντηση δίνεται στο κεφάλαιο των λύσεων.

3.1.3 Γραφικά

Όπως σας υποσχεθήκαμε νωρίτερα, θα σας παρουσιάσουμε τώρα μερικές εντολές γραφικών του Amstrad. Όπως θα ανακαλύψατε νωρίτερα το μέγεθος των χαρακτήρων στην οθόνη ποικίλει σε κάθε τρόπο. Γίνονται μέγιστοι στο MODE 2 και μικροσκοπικοί στο MODE 0. Μια παρόμοια κατάσταση υπάρχει όταν χρησιμοποιείτε γραφικά. Για να παράγουμε σχέδια χρειαζόμαστε να "Φωτίσουμε" τα κατάλληλα σημεία πάνω στην οθόνη. Τα σημεία αυτά είναι γνωστά σαν "PIXELS" (συντομογραφία του Picture Elements <στοιχεία εικόνας> κατά κάποιον τρόπο!) και ακόμα ποικίλουν στο μέγεθος με κάθε τρόπο (μέγιστα στο MODE 2 και ελάχιστα στο MODE 0). Η διαφορά στο μέγεθος είναι γνωστή σαν διακριτικότητα δηλαδή η λεπτομέρεια της εμφάνισης σε κάθε τρόπο (mode). Αν παρατηρήσετε από κοντά μια φωτογραφία σε μια εφημερίδα θα δείτε ότι αποτελείται από εκατοντάδες μικρές τελείες, το ίδιο συμβαίνει και με τα γραφικά στον Amstrad. Όσο μικρότερο είναι το στοιχείο, τόσο καθαρότερη και η εμφάνιση που θα πάρετε. Η διαφορά στο μέγεθος του στοιχείου απεικονίζεται στο σχήμα 3.5.

MODE	PIXEL SIZE (relative)	PIXELS PER SCREEN
0		160 x 200 pixels
1		320 x 200 pixels
2		640 x 400 pixels

Εικόνα 3.5

3.1.3.1 PLOT

Κάθε σημείο έχει τις δικές του μοναδικές συντεταγμένες "X" και "Y". Αυτές διακυμαίνονται από 0,0 (η κάτω αριστερή γωνία) ως 640,400 (η πάνω δεξιά γωνία). Για να 'τοποθετήσετε' δηλαδή να φωτίσετε ένα στοιχείο σε οποιοδήποτε σημείο της οθόνης, χρησιμοποιείτε την εντολή PLOT με τη μορφή:

```
PLOT X,Y
```

όπου X και Y είναι οι συντεταγμένες του στοιχείου που θα τοποθετηθεί. Για παράδειγμα:

```
PLOT 100,100
```

Αυτό θα φωτίσει ένα στοιχείο στο σημείο 100 κατά μήκος και 100 κατά πλάτος από το αρχικό. Το αρχικό είναι που έχει τη συντεταγμένη τιμή 0,0 δηλαδή η αριστερή κάτω γωνία της οθόνης. Το χρώμα αποφασίζεται από την τιμή INK 1.

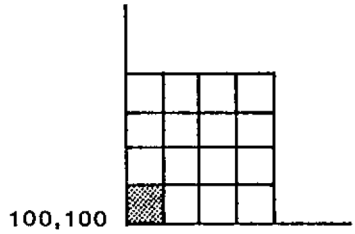
Για να κάνετε PLOT ένα στοιχείο στο κέντρο της οθόνης, πληκτρολογήστε το ακόλουθο πρόγραμμα:

ΠΡΟΓΡΑΜΜΑ 3.1

```
10 PLOT 320,200
```

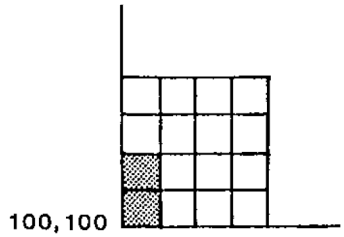
'Οταν σχεδιάζετε ένα σημείο ο Amstrad βλέπει δύο πράγματα. Πρώτα τις συντεταγμένες X,Y και δεύτερον το MODE. Στο MODE 2 οι συντεταγμένες των στοιχείων (PIXEL) αντιστοιχούν ακριβώς με τις συντεταγμένες σημείων,

επομένως στο MODE 2 ο Amstrad θα σχεδιάσει μόνο το σημείο που καθορίστηκε από τις συντεταγμένες. Αυτό παρουσιάζεται παρακάτω:



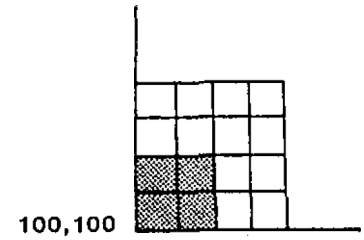
Εικόνα 3.6(α) Ένα σημείο στο Mode 2.

Ωστόσο όταν χρησιμοποιείται το MODE 1 ο υπολογιστής, σχεδιάζει δύο σημεία γιατί κάθε σημείο (PIXEL) στο MODE 1 είναι διπλάσιο απ' αυτά στο MODE 2. Στο MODE 1 τα σημεία είναι ομαδοποιημένα μαζί σε δυάδες. Συνεπώς η σχεδίαση σε μια θέση θα προκαλέσει τη σχεδίαση του δευτέρου στην ομάδα. Έτσι η σχεδίαση της θέσης 100,100 θα προκαλέσει το άναμα της θέσης 100,101 επίσης.



Εικόνα 3.6(β) Ένα σημείο στο Mode 1.

Η μορφή MODE 0 μεταχειρίζεται κατά τον ίδιο ακριβώς τρόπο, αλλά τη φορά αυτή το μέγεθος του στοιχείου είναι διπλάσιο από αυτό του MODE 1 και τετραπλάσιο από αυτό του MODE 2. Συνεπώς μια εντολή σχεδίου στη μορφή MODE 0 προκαλεί την τοποθέτηση των 4 σημείων.



Εικόνα 3.6(γ) Ένα σημείο στο Mode 0.

Με άλλα λόγια τα μεγέθη των στοιχείων μπορεί να διαφέρουν αλλά, ανεξάρτητα από τη μορφή που χρησιμοποιείται, η οθόνη των γραφικών έχει πάντοτε τον ίδιο αριθμό γραφικών σημείων δηλαδή 640x400. Αυτό σημαίνει ότι όταν χρησιμοποιούμε τα MODE 0 ή 1 (όπου η διάστιξη είναι λιγότερη από 640x400), σχεδιάζοντας σε ένα οποιοδήποτε συγκεκριμένο γραφικό σημείο πάνω στην οθόνη, θα "ανάψουν" ορισμένα σημεία που το περιβάλλουν. Για να επιδείξουμε αυτό (?) παραπέρα, πληκτρολογήστε και τρέξετε το πρόγραμμα 3.2.

ΠΡΟΓΡΑΜΜΑ 3.2

```
5 MODE 0
10 FOR X=0 TO 640 STEP 4
20 PLOT X,100
30 NEXT X
```

Επειδή κάθε σημείο στο MODE 0 παίρνει τέσσερα (4) σημεία, το πρόγραμμα αυτό θα παράγει μια ολόκληρη σειρά πάνω στην οθόνη ακόμα και αν είναι να σχεδιάσετε τέσσερα - τέσσερα (είναι ακόμα πιο γρηγοράτερο). Όταν τελειώσει αλλάξετε τη γραμμή 5 στο ακόλουθο:

```
5 MODE 1
```

Τώρα τρέξετε το πρόγραμμα. Αυτή τη φορά επειδή τα σημεία του MODE 1 είναι μισά στο μέγεθος από τα σημεία του MODE 0, το πρόγραμμα θα παράγει μια διακεκομένη γραμμή, όπου τοποθετείται κάθε δεύτερο σημείο. Όταν το πρόγραμμα τελειώσει πάλι, αλλάξετε τη γραμμή 5.

```
5 MODE 2
```

Τη φορά αυτή κάθε τέταρτο σημείο τοποθετείται μια ακομα πιο διακεκομένη γραμμή.

ΜΕΡΟΣ ΔΕΥΤΕΡΟ

3.2.1 Ένα παιχνίδι χάραξης – σχεδίου

Το κύριο πρόβλημα που έχει σχέση με κάθε τύπο παιχνιδιού χάραξης – σχεδίου, είναι ότι το πρόγραμμα θα πρέπει να κάνει τον υπολογιστή να σχεδιάζει γραμμές ακαθόριστου μήκους.

Το πρόβλημα αυτό μπορεί να προσπεραστεί τακτοποιώντας το έτσι ώστε να αυξάνει ελαφρά το μήκος μιας γραμμής κάθε φορά που πιέζεται το πλήκτρο. Το πρόβλημα αυτό αντιμετωπίζεται στο πρόγραμμα 3.3. Για να ετοιμάσουμε τον Amstrad να δεχθεί ένα νέο πρόγραμμα είναι απαραίτητο να πληκτρολογήσετε την εντολή NEW. Κάντε το τώρα και μετά πληκτρολογήσετε το πρόγραμμα 3.3.

ΠΡΟΓΡΑΜΜΑ 3.3

```
10 MODE 1
70 INPUT D$
80 PLOT X,Y
90 X=X+1
100 GOTO 70
```

Αν παρακολουθήσετε το πρόγραμμα, να τι κάνει:

* Πρώτα η οθόνη καθαρίζεται και τοποθετείται στο MODE 1.

* Ο υπολογιστής ρωτά για ένα εισαγόμενο (δεδομένο εισόδου) και περιμένει να πατηθεί το ENTER.

* Το σημείο (X,Y) συνδιάζεται στην οθόνη (Μπορεί να πρέπει να κοιτάξετε προσεκτικά για αυτό!).

* Η συντεταγμένη X αυξάνεται κατά 1.

* Το πρόγραμμα ανακυκλώνεται στη γραμμή 70 για ένα άλλο εισαγόμενο. Το σημείο του ερωτηματικού εμφανίζεται απότομα μια γραμμή χαμηλότερα κάθε φορά που φτάνεται η γραμμή 90.

* Μετά από ένα δεύτερο εισαγόμενο τοποθετείται άλλο ένα σημείο. Δεν μπορείτε να δείτε αυτό το σημείο γιατί οι συντεταγμένες της PLOT έχουν ήδη τοποθετηθεί από την πρώτη PLOT πρόταση. Με τη

μέθοδο αυτή θα χρειασθούν δύο εισαγόμενα για τον καθορισμό κάθε σημείου.

Για να δεις ένα νέο σημείο μετά από κάθε εισαγόμενο, η συντεταγμένη X θα πρέπει να αυξηθεί ανά δύο. Αυτό ωστόσο, είναι σωστό μόνο για το MODE 1. Όσοι από εσάς θα επιθυμήσουν να δοκιμάσουν το πρόγραμμα στο MODE 1 ή 2, θα πρέπει να προσθέσουν το 4 και το 1 στο X αντίστοιχα.

Αυτό αφήνει ακόμα ένα πρόβλημα. Το αντικείμενο του παιχνιδιού είναι να αυξάνεται η γραμμή κάθε φορά που πατιέται ένα πλήκτρο, αλλά χρησιμοποιώντας την INPUT πρέπει να πατήσετε και το ENTER επίσης. Όχι μόνο αυτό αλλά και το ερωτηματικό εμφανίζεται αμέσως πάνω στην οθόνη. Υπάρχει μια εντολή που σας επιτρέπει να εισάγετε χαρακτήρες χωρίς να πατήσετε το ENTER και χωρίς να πρέπει να εμφανίζεται καμμία μορφή εισαγομένων απότομα. Αυτή είναι η εντολή INKEY$.

3.2.1.1 INKEY$

Το τι κάνει η εντολή INKEY$, όπως δηλώνει και το όνομά της, είναι να παίρνει μέσα (IN) την τιμή από όποιο πλήκτρο πληκτρολογείται. Η εντολή παίρνει τη μορφή:

```
A$=INKEY$
```

όπου το A$ είναι μια αλφαριθμητική μεταβλητή (string) στην οποία θέλετε να αποθηκευτεί η τιμή του πατημένου πλήκτρου.

Οι σημαντικές διαφορές μεταξύ του INKEY$ και του INPUT είναι ότι η INKEY$ δέχεται το κτύπημα του πλήκτρου χωρίς να χρειάζεται να πατηθεί το ENTER και ακόμα ο χαρακτήρας δεν εμφανίζεται στην οθόνη. Όπότε απαντήσει την INKEY$ ο υπολογιστής εξερευνά την περιοχή της μνήμης όπου είναι αποθηκευμένοι οι χαρακτήρες από τα πατημένα πλήκτρα. Αυτό είναι γνωστό σαν ο "προσωρινός καταχωρητής πληκτρολογίου" (buffer). Ο πρώτος χαρακτήρας που βρίσκει καταχωρείται στην A$ και αν δε βρεθεί τίποτα, τότε μεταβιβάζεται στο A$ η κενή αλφαριθμητική και το πρόγραμμα απλώς πάει παρακάτω. Η εντολή αυτή μπορεί να χρησιμοποιηθεί για να σταματήσει προσωρινά το πρόγραμμα, μέχρι να πιεστεί ένα πλήκτρο δοκιμάζοντας απλά τον προσωρινό καταχωρητή του πληκτρολογίου για να δούμε αν ένα συγκεκριμένο πλήκτρο έχει πατηθεί. Αν η δοκιμή αποδειχθεί αρνητικά, δηλαδή αν το πλήκτρο δεν έχει πατηθεί, τότε το πρόγραμμα επιστρέφει στην αρχή της γραμμής και ο έλεγχος επαναλαμβάνεται. Το ακόλουθο πρόγραμμα δείχνει αυτό.

ΠΡΟΓΡΑΜΜΑ 3.4(a)

```
2 PRINT "PRESS 8 TO BEGIN"
3 A$=INKEY$
4 IF A$ <>"8" THEN 3
```

Όταν το πρόγραμμα τρέξει τώρα, αρχικά δε συμβαίνει τίποτα, μέχρι να πατηθεί το πλήκτρο "8". Μόλις γίνει αυτό το πρόγραμμα συνεχίζει να λειτουργεί όπως έκανε προηγουμένως.

Με μια γραμμή όπως η 4 στο πρόγραμμα 3.4(a) το πρόγραμμα περιμένει μέχρι να πατηθεί ένα συγκεκριμένο πλήκτρο. Για να λειτουργήσει ικανοποιητικά αυτή η γραμμή του προγράμματος, ο χρήστης χρειάζεται να του έχουν πει ποιό πλήκτρο πρέπει να πατήσει. Ένα πρόγραμμα μπορεί να γίνει πιο γενικό χρησιμοποιώντας την εντολή INKEY$ να κοιτάζει στον καταχωρητή για το αν δεν έχει πατηθεί κανένα πλήκτρο στην περίπτωση που η "κενή αλφαριθμητική" θα αποθηκευτεί. Στη BASIC μια κενή αλφαριθμητική μεταβλητή περιγράφεται με δύο διπλά εισαγωγικά μαζί δηλαδή "". Σ' ένα πρόγραμμα θα εισαγόταν όπως στη γραμμή 4 του παρακάτω προγράμματος 3.4(b). Η γραμμή αυτή λέει ότι αν ο καταχωρητής περιέχει την κενή αλφαριθμητική, δηλαδή τίποτα, τότε πηγαίνετε πίσω για ένα άλλο εισαγόμενο. Η διαδικασία αυτή θα επαναλαμβάνεται για πάντα ωσότου να πατηθεί ένα κουμπί, εκτός από το SHIFT ή το CAPS LOCK. Και το πλήκτρο ESC ισχύει επίσης, αλλά αυτό μόνο σταματάει για λίγο το πρόγραμμα, ως συνήθως.

ΠΡΟΓΡΑΜΜΑ 3.4(b)

```
4 IF A$="" THEN 3
```

Ένα τέτοιο τέχνασμα χρησιμοποιείται συνήθως όταν κάποιος χρειάζεται να σταματήσει ένα πρόγραμμα ενώ ο χρήστης θα διαβάζει ένα μήνυμα. Στη περίπτωση αυτή κάτι χρήσιμο ενσωματώνεται για να πει στο χειριστή τι αναμένεται. Για να γίνει αυτό, οι γραμμές 2 και 6 του προγράμματος 3.4(c) πρέπει να προστεθούν. Η γραμμή 2 εκτυπώνει το μήνυμα πάνω στην οθόνη και μόλις πατηθεί ένα πλήκτρο η γραμμή 6 καθαρίζει την οθόνη χρησιμοποιώντας την εντολή CLS.

ΠΡΟΓΡΑΜΜΑ 3.4(c)

```
2 PRINT"PRESS ANY KEY TO BEGIN"
3 A$=INKEY$
4 IF A$="" THEN 3
6 CLS
```

Οι γραμμές 2 ως 6 χρησιμοποιήθηκαν μόνο για σκοπούς επίδειξης και δε χρειάζονται άλλο, έτσι θα πρέπει να απαλλαχθούμε από αυτές. Αυτό προσφέρει την ευκαιρία να εξετάσουμε άλλη μια εντολή ελέγχου της Amstrad BASIC.

3.2.1.2 DELETE

Για να μετακινήσουμε μια μόνο γραμμή πληκτρολογήσετε μόνο το σχετικό αριθμό γραμμής και μετά πατήστε ENTER. Αυτή η διαδικασία μπορεί να επαναληφθεί μέχρι να αφαιρεθούν όλες οι ανεπιθύμητες γραμμές. Ωστόσο, η Amstrad BASIC έχει έναν ευκολότερο τρόπο να αφαιρεί ομάδες από διαδοχικές γραμμές προγράμματος. Αυτός είναι η εντολή DELETE.

Για να αφαιρέσετε μια ολόκληρη ομάδα γραμμών πρέπει να καθορίσετε πρώτα δύο πράγματα. Πρώτα, τον αριθμό της γραμμής που θέλετε να ξεκινήσει η διαγραφή και δεύτερον, τον αριθμό της γραμμής στην οποία θα τελειώσει η διαγραφή. Έτσι για να διαγράφετε τις γραμμές 2 ως 6 πληκτρολογείστε:

```
DELETE 2-6
```

Μόλις πατήσετε το ENTER, οι γραμμές αυτές θα διαγραφούν. Δεν υπάρχουν πια! Πρέπει να είστε προσεκτικοί όταν χρησιμοποιείτε την εντολή DELETE. Αν κάνετε ένα λάθος θα μπορούσατε να καταλήξετε να έχετε μετακινήσει ολόκληρο το πρόγραμμα.

Η DELETE έχει διαφοροποιήσεις όμοιες μ' εκείνες που χρησιμοποιούνται με τη LIST.

`DELETE` : Διαγράφει όλο το πρόγραμμα

`DELETE 10` : Διαγράφει μόνο τη γραμμή 10

`DELETE 10-100` : Διαγράφει όλες τις γραμμές από 10-100 συμπεριλαμβανόμενες (10,100)

`DELETE 10-` : Διαγράφει όλες τις γραμμές από 10 και πέρα.

DELETE -100 : Διαγράφει όλες τις γραμμές μέχρι και της γραμμής 100 συμπεριλαμβανόμενης

Επιστρέφοντας στο παιχνίδι χάραξης - σχεδίου, η εντολή INKEY$ μπορεί να ενσωματωθεί μέσω της γραμμής 70. Αν δεν έχει πατηθεί κανένα πλήκτρο, τότε το πρόγραμμα θ' ανακυκλωθεί "εσωτερικά" στη γραμμή 70 μέχρι να πατηθεί ένα.

ΠΡΟΓΡΑΜΜΑ 3.5

```
70 A$=INKEY$:IF A$="" THEN 70
```

Η επόμενη δουλειά είναι να τροποποιήσουμε το πρόγραμμα έτσι ώστε να σχεδιάζει μια γραμμή σε μια επιλεγμένη κατεύθυνση. Αυτό πετυχαίνεται δοκιμάζοντας την τιμή της A$. Το πληκτρολόγιο του Amstrad έχει 4 κατευθυντήρια πλήκτρα με βέλη, να δείχνουν σε κάθε κατεύθυνση (αριστερά, δεξιά, πάνω και κάτω). Θα ήταν ωραίο να μπορούσαμε να χρησιμοποιήσουμε αυτά τα πλήκτρα για να μετακινήσουμε τη "γραμμή" στην κατάλληλη κατεύθυνση. Για να το πετύχουμε, το πρόγραμμα θα έπρεπε να περιέχει τις παρακάτω σειρές οδηγιών:

```
IF A$="↑" THEN Y=Y+1
IF A$="↓" THEN Y=Y-1
IF A$="←" THEN X=X-2
IF A$="→" THEN X=X+2
```

Ωστόσο, υπάρχει ένα μικρό πρόβλημα. Όταν επιχειρείτε να πληκτρολογήσετε ένα βέλος ανάμεσα στα εισαγωγικά, δεν παίρνετε βέλος, αλλά ο δρομέας μετακινείται στην κατεύθυνση του βέλους. Αυτό είναι πράγματι ένα πρόβλημα. Αν τα βέλη του δρομέα δεν εμφανιστούν μέσα στα εισαγωγικά πως μπορεί να δοκιμαστεί ένα εισαγόμενο για αυτά; Η βοήθεια είναι στο χέρι.

3.2.1.3 CHR$() και ASC()

Κάθε πλήκτρο στο πληκτρολόγιο του Amstrad έχει μια ειδική αριθμητική τιμή που αναφέρεται σαν η ASCII τιμή του. Τα ASCII είναι τα αρχικά από το American Standard Code for Information Interchange. Ο τρόπος για να βρείτε την ASCII τιμή του κάθε χαρακτήρα είναι να τυπώσετε την τιμή καθενός ASCII χρησιμοποιώντας τη συνάρτηση ASCII. Για παράδειγμα:

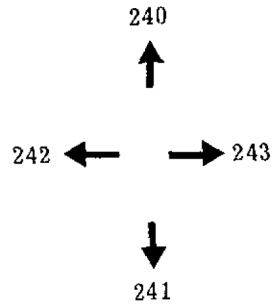
```
PRINT ASC("A")
```

Θα έχει σαν αποτέλεσμα την εμφάνιση του 65. Το 65 είναι η ASCII τιμή του κεφαλαίου "A". Το γράμμα "A" μπορεί τώρα να εμφανιστεί στην οθόνη πληκτρολογώντας το CHR$(65); π.χ.

```
PRINT CHR$(65)
```

Ο αριθμός που περικλείεται στις παρενθέσεις είναι μια ASCII τιμή ενός χαρακτήρα, στην περίπτωση αυτή του κεφαλαίου "A".

Έτσι τα πλήκτρα με τα βέλη μπορούν να δοκιμαστούν χρησιμοποιώντας τις ASCII τιμές τους που είναι



Η δοκιμή μπορεί να γίνει μ' έναν από τους δύο τρόπους:

```
IF ASC(241)=A$ THEN
```

ή

```
IF A$=CHR$(241) THEN
```

Η τελευταία έχει επιλεχθεί αλλά αν προτιμάτε να χρησιμοποιήσετε την άλλη μπορείτε να το κάνετε.

ΠΡΟΓΡΑΜΜΑ 3.5(a)

```
80 IF A$=CHR$(240) THEN Y=Y+1
90 IF A$=CHR$(241) THEN Y=Y-1
100 IF A$=CHR$(242) THEN X=X-2
110 IF A$=CHR$(243) THEN X=X+2
220 PLOT X,Y
230 GOTO 70
```

Όταν αυτές οι γραμμές προστεθούν, το πρόγραμμα θα πρέπει

να σχεδιάζει σ' όλες τις 4 κατευθύνσεις. Πειραματιστείτε για να δείτε αν μπορείτε να σχεδιάσετε ορθογώνια χρησιμοποιώντας την εντολή.

Υπάρχει ωστόσο ακόμα μια ατέλεια. Εδώ το πρόγραμμα δεν θα σταματήσει εκτός αν πατήσετε το ESC δύο φορές. Επομένως θα προσθεθεί μια παραπάνω INKEY$ δοκιμή. Τη φορά αυτή αν η δοκιμή είναι θετική, δίνεται η εντολή STOP.

ΠΡΟΓΡΑΜΜΑ 3.5(b)

```
120 IF A$="S" THEN STOP
```

Αυτό το πρόγραμμα δουλεύει τώρα σίγουρα, αλλά υπάρχουν ακόμα "παράσιτα". Για παράδειγμα, είναι δυνατόν ακόμα να σχεδιάσετε και έξω από τις άκρες της οθόνης. Ο Amstrad θα σας επιτρέψει να συνεχίσετε να σχεδιάζετε σημεία και έξω από την οθόνη, αλλά αυτό δεν είναι πολύ διασκεδαστικό καθώς δεν θα μπορείτε να δείτε τίποτα. Ένας τρόπος για να τ' αποφύγετε αυτό είναι να χρησιμοποιήσετε περισσότερες σειρές δοκιμών, όπως φαίνονται παρακάτω στο πρόγραμμα 3.5(c):

ΠΡΟΓΡΑΜΜΑ 3.5(c)

```
130 IF X>640 THEN X=640
140 IF X<0 THEN X=0
150 IF Y>400 THEN Y=400
160 IF Y<0 THEN Y=0
```

Αν συμπεριλάβουμε αυτές τις γραμμές τότε θα είναι αδύνατο να σχεδιαστούν σημεία έξω από την οθόνη, όπως για παράδειγμα όταν το X υπερβαίνει το 640, η γραμμή 100 το ξαναποθετεί στη μέγιστη αυτή τιμή. Οι γραμμές 110, 120 και 130 κάνουν μια παρόμοια εξυπηρέτηση για την ελάχιστη τιμή του X και τη μέγιστη και ελάχιστη τιμή του Y αντίστοιχα. Ολόκληρο το πρόγραμμα καταγράφεται παρακάτω για τη δική σας ευκολία. Οι γραμμές 20 έως 60 έχουν προστεθεί για να επιτρέψουν στο χρήστη να διαλέξει τη θέση από την οποία θα ξεκινήσει το σχεδίασμα:

ΠΡΟΓΡΑΜΜΑ 3.5(d)

```
10 MODE 1
20 LOCATE 1,1
40 INPUT"X START";X
50 INPUT"Y START";Y
60 PLOT X,Y
70 A$=INKEY$
80 IF A$=CHR$(240) THEN Y=Y+1
90 IF A$=CHR$(241) THEN Y=Y-1
100 IF A$=CHR$(242) THEN X=X-2
110 IF A$=CHR$(243) THEN X=X+2
120 IF A$="S" THEN STOP
130 IF X>640 THEN X=640
140 IF X<0 THEN X=0
150 IF Y>400 THEN Y=400
160 IF Y<0 THEN Y=0
220 PLOT X,Y
230 GOTO 70
```

ΑΣΚΗΣΗ 3.2

Προσθέστε μία επιπλέον γραμμή έτσι που αν πατηθεί το πλήκτρο "R" το πρόγραμμα να επιτρέψει τη σχεδίαση να αρχίσει πάλι σε καινούρια θέση αρχής. Μία δυνατή λύση δίνεται στο κεφάλαιο των λύσεων.

ΜΕΡΟΣ ΤΡΙΤΟ

3.3.0.1 DRAW

Η PLOT δε βάζει μόνο ένα σημείο πάνω στην οθόνη, αλλά ακόμη μετακινεί το δρομέα των γραφικών στις συντεταγμένες του σχεδίου. Ο δρομέας γραφικών χρησιμοποιείται, για να παρακολουθεί τα ίχνη από το τελευταίο σημείο που αποτυπώθηκε από τον υπολογιστή. Οι συντεταγμένες αυτές είναι οι αρχικές τιμές για την εντολή DRAW.

ΠΡΟΓΡΑΜΜΑ 3.6

```
10 MODE 1
20 PLOT 50,25
```

Το πρόγραμμα αυτό θα θέσει τον υπολογιστή στο MODE 1 και μετά θα σχεδιάσει ένα σημείο στις συντεταγμένες 50,25. Θα έχετε την ικανότητα να δείτε αυτό το σημείο αν κοιτάζετε από αρκετά κοντά, είναι στο κάτω αριστερό μέρος της οθόνης. Αυτό είναι το σημείο αφετηρίας για το σχεδίασμα της εντολής DRAW.

ΠΡΟΓΡΑΜΜΑ 3.6(a)

```
30 DRAW 100,100
```

Αυτό θα προκαλέσει τη σχεδίαση μιας γραμμής από τον Amstrad από το σημείο 50,25 (όπως προσδιορίστηκε από την εντολή PLOT) στο σημείο 100,100. Η εντολή DRAW μας επιτρέπει να σχεδιάζουμε γραμμές, ξεκινώντας από τις συντεταγμένες που διαλέχτηκαν με την εντολή PLOT και τελειώνοντας στις συντεταγμένες που καθορίστηκαν από την ίδια την εντολή DRAW.

Το χρώμα της σχεδιασμένης γραμμής προκαθορίζεται στη τιμή της INK 1. Αυτή μπορεί να αλλαχθεί προσθέτοντας τον αριθμό μελάνης που περιέχει το απαιτούμενο χρώμα για την εντολή DRAW.

ΠΡΟΓΡΑΜΜΑ 3.6(b)

```
30 DRAW 100,100,3
```

Κοιτάζοντας το σχήμα 3.4 στη σελίδα 3.4 βλέπουμε ότι η INK 3 έχει τη τιμή του κόκκινου χρώματος ενώ είναι στο MODE 1.

ΑΣΚΗΣΗ 3.3

Ζωγραφίστε μία γαλαζοπράσινη γραμμή από το κέντρο της οθόνης στη θέση 64,93 στη MODE 0. Η απάντηση υπάρχει στο κεφάλαιο των λύσεων.

Η εντολή DRAW μετακινεί μόνη της το δρομέα των γραφικών, έτσι η επόμενη εντολή DRAW θα συνεχίσει από όπου σταμάτησε η τελευταία.

ΠΡΟΓΡΑΜΜΑ 3.6(c)

```
10 MODE 1
20 PLOT 50,25
30 DRAW 100,100
40 DRAW 200,25
```

Η γραμμή 20, τοποθετεί το δρομέα των γραφικών στις συντεταγμένες 50,25 δηλαδή 50 σημεία κατά πλάτος και 25 προς τα πάνω. Η εντολή DRAW στη γραμμή 30 ξεκινά από το σημείο αυτό και σχεδιάζει μια γραμμή στο εκατοστό σημείο κατά πλάτος και εκατοστό σημείο προς τα πάνω. Το σημείο αυτό χρησιμοποιείται από την επόμενη DRAW, γραμμή 40, καθώς το σημείο αφετηρίας και η γραμμή σχεδιάζεται από τις συντεταγμένες 100,100 στις 200,25.

3.3.0.2 DRAWR

Η διαφοροποίηση αυτής της εντολής DRAW σας επιτρέπει να ΣΧΕΔΙΑΣΕΤΕ μια γραμμή σε ένα σημείο ΣΧΕΤΙΚΟ με την τρέχουσα θέση του δρομέα των γραφικών. Για παράδειγμα:

```
PLOT 300,100:DRAW 50,50
```

Αυτό θα σχεδιάσει μια γραμμή ξεκινώντας από το 300,100 και τελειώνοντας στο σημείο 50,50. Η DRAWR ωστόσο, θα κάνει κάτι κάπως διαφορετικό.

```
PLOT 300,100:DRAWR 50,50
```

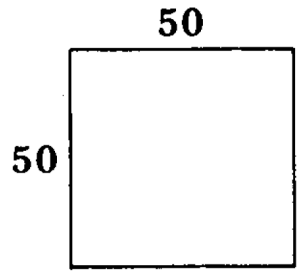
Αυτό θα σχεδιάσει μια γραμμή από το σημείο 300,100 στη θέση 350,150, μια θέση 50 σημεία πάνω και κατά πλάτος από την θέση αφετηρίας.

Χρησιμοποιώντας τις PLOT και DRAWR εντολές είναι δυνατό να γράφετε ένα μικρό πρόγραμμα που επιτρέπει στο χρήστη να σχεδιάσει μια ποικιλία κουτιών πάνω στην οθόνη. Το πρόγραμμα 3.7 σχεδιάζει ένα τετράγωνο με πλευρές 50 σημείων.

ΠΡΟΓΡΑΜΜΑ 3.7

```
10 MODE 1
20 PLOT 320,200
30 DRAWR 50,0
40 DRAWR 0,-50
50 DRAWR -50,0
60 DRAWR 0,50
```

Η γραμμή 30 σχεδιάζει ένα σημείο στο κέντρο της οθόνης. Μετά χρησιμοποιώντας την DRAWR, η γραμμή 30 σχεδιάζει μια γραμμή 50 σημεία προς τα δεξιά. Αυτή είναι η ψηλότερη γραμμή του τετραγώνου μας. Το επόμενο στάδιο είναι να σχεδιάσει τη δεξιά πλευρά (γραμμή 40). Έχοντας μια συντεταγμένη X με τιμή μηδέν στην εντολή DRAWR σημαίνει ότι η γραμμή θα σχεδιαστεί χρησιμοποιώντας την ίδια συντεταγμένη X. Έχοντας μια αρνητική τιμή σαν συντεταγμένη Y κάνει τον υπολογιστή να σχεδιάσει τη γραμμή κάτω στην οθόνη. Το πρόγραμμα μέχρι τη γραμμή 40 παράγει την ακόλουθη εικόνα.



Εικόνα 3.7

Οι γραμμές 50 και 60 συμπληρώνουν το τετράγωνο σχεδιάζοντας πρώτα το κάτω μέρος και μετά την αριστερή πλευρά.

Το πρόγραμμα αυτό, αν και όμορφο είναι βαρετό. Σχεδιάζει πάντα το ίδιο κουτί. Πραγματικά θα θέλαμε να παράγουμε πολλά διαφορετικά κουτιά, μερικά απ' αυτά τετράγωνα και μερικά ορθογώνια. Για να το κάνουμε αυτό, οι συντεταγμένες σχεδίασης θα πρέπει να'ναι μεταβλητές. Στο ξεκίνημα του προγράμματος ο χρήστης πρέπει να ερωτηθεί να εισαγάγει τις συντεταγμένες και για τη X και για τη Y. Προσθέσετε γραμμές 2 και 4 του 3.7(a) στο 3.7:

ΠΡΟΓΡΑΜΜΑ 3.7(a)

```
2 INPUT"TOP LEFT X:";X
4 INPUT"TOP LEFT Y:";Y
```

Τώρα η γραμμή 20, χρειάζεται να προσαρμοστεί ανάλογα.

ΠΡΟΓΡΑΜΜΑ 3.7(b)

```
20 PLOT X,Y
```

Το πρόγραμμα θα σχεδιάσει τώρα το ίδιο κουτί οπουδήποτε στην οθόνη. Για να κάνουμε το πρόγραμμα πραγματικά γενικό, θα έπρεπε να εισαγάγουμε τις τιμές του ύψους και του πλάτους, ενός κουτιού. Αυτές οι τροποποιήσεις περιέχονται στο πρόγραμμα προσθέτοντας σε αυτό το πρόγραμμα 3.7(c).

ΠΡΟΓΡΑΜΜΑ 3.7(c)

```
6 INPUT"HEIGHT OF BOX:";H
8 INPUT"WIDTH OF BOX :";W

30 DRAWR W,0
40 DRAWR 0,-H
50 DRAWR -W,0
60 DRAWR 0,H
```

Η ρουτίνα σχεδίασης-κουτιού, είναι τώρα "καθολική" και μπορούμε να σχεδιάσουμε ένα ορθογώνιο σχήμα οπουδήποτε πάνω στην οθόνη.

ΑΣΚΗΣΗ 3.4

Προσθέστε τις γραμμές 3 και 5 στο πρόγραμμα 3.7(a-c) έτσι ώστε η θέση αρχής για το κουτί να ελεγχθεί για να δείτε εάν είναι μέσα στα επιτρεπόμενα όρια. Εάν δεν είναι τότε το πρόγραμμα θα ξαναρχίσει, για να λάβει μία άλλη είσοδο. Μία δυνατή απάντηση δίνεται στο κεφάλαιο των λύσεων.

ΜΕΡΟΣ ΤΕΤΑΡΤΟ

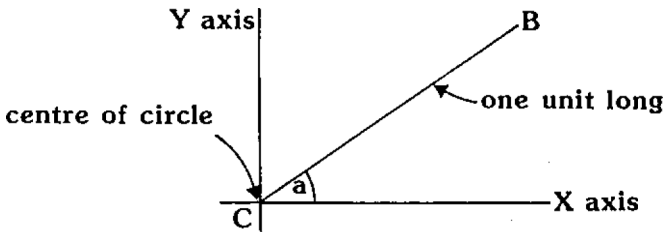
3.4.1 Κύκλοι

Ο Amstrad δεν έχει εντολή για κύκλο, αλλά είναι δυνατό να σχεδιάσετε ένα κύκλο χρησιμοποιώντας τα PLOT. Αν μπορούσε να βρεθεί κάποιος τρόπος να υπολογίσουμε τις θέσεις σχεδίασης ενός κύκλου, τότε θα μπορούσε να "σχεδιαστεί" ένας κύκλος στην οθόνη. Η εργασία αυτή δεν είναι τόσο αδύνατη όσο ακούγεται γιατί η βοήθεια είναι στο χέρι.

3.4.1.1 SIN() και COS()

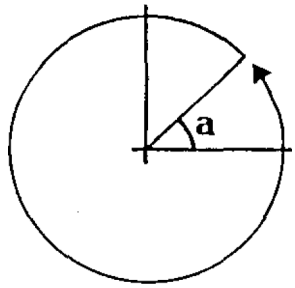
Για έναν κύκλο, οι μαθηματικές συναρτήσεις που χρειαζόμαστε, είναι οι παλιοί μας φίλοι από το σχολείο, SIN και COS. Αυτές οι δύο χρήσιμες συναρτήσεις θα χρησιμοποιηθούν πάρα πολύ όταν σχεδιάζουμε ένα κύκλο. Ας δούμε πρώτα τι είναι και πως μπορούν να βοηθήσουν στην εργασία αυτή.

Το σχήμα 3.8 δείχνει τους άξονες X και Y (οριζόντιας και κάθετης κατεύθυνσης) με μια γραμμή "CB" με μήκος μιας μονάδας, σχεδιασμένη από το κέντρο "C" του κύκλου, κατά μια γωνία "a" από τον άξονα "X".



Εικόνα 3.8

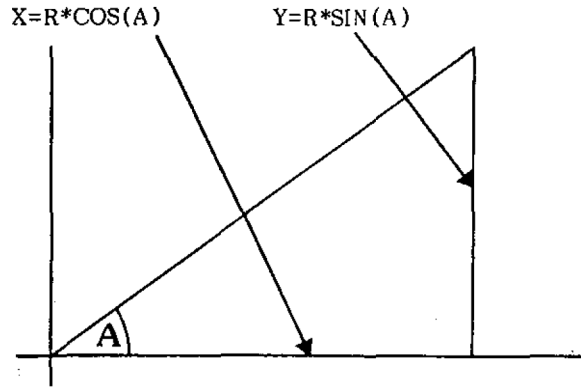
Αν η γραμμή CB περιστρεφόταν με κέντρο το C και τελικό σημείο το B θα μπορούσε να χαράξει ένα κύκλο.



Εικόνα 3.9

Για να χρησιμοποιήσουμε αυτές τις ιδέες για να σχεδιάσουμε ένα κύκλο στον υπολογιστή θα χρειαστεί να ξέρουμε τις συντεταγμένες "X" και "Y" του τελικού σημείου CB. Αυτό, όπως θα έχετε μαντέψει, είναι το σημείο που εισάγονται τα SIN και COS.

Για μια συγκεκριμένη γωνία "A" οι συντεταγμένες "X" και "Y" είναι:



Εικόνα 3.10

Η τιμή αυτή "A" θα μπορούσε να είναι σε ακτίνια, δηλαδή 2*π ακτίνια ή σε 360 μοίρες (ένας κύκλος). Το π (προφέρεται "π" όπως μηλόπιττα) είναι ένας αριθμός που είναι ο λόγος της περιφέρειας του κύκλου προς την ακτίνα του – η περίμετρος ενός κύκλου είναι 2*π*R (R=ακτίνα). Η πραγματική τιμή του π είναι 3.14159... αλλά δε χρειάζεται να ανησυχείτε γι' αυτό γιατί ο Amstrad έχει μια ειδική μεταβλητή που καλείται PI. Αυτή έχει την τιμή 3.14159265. Ελέγξετε το πληκτρολογώντας:

```
PRINT PI
```

Ο Amstrad σας δουλεύει σε ακτίνια γιατί η μαθηματική φόρμουλα που χρησιμοποιεί ο υπολογιστής για να υπολογίσει το COS και SIN δουλεύει φυσιολογικά σε ακτίνια.

Αν βάλουμε την "A" να είναι όλη σε γωνίες σε έναν κύκλο δηλαδή 0 ως 2*PI ακτίνια, τότε η SIN(A) και COS(A) δίνουν όλες τις συντεταγμένες "X" και "Y" των σημείων γύρω από τον κύκλο. Καλά, όλα αυτά είναι πολύ καλά για έναν κύκλο με μια ακτίνα "1", αλλά τι θα γίνει με έναν κύκλο με ακτίνα "100"; Αν η ακτίνα είναι 100 φορές μακρύτερη, τότε όλα είναι 100 φορές μεγαλύτερα, έτσι οι συντεταγμένες "X" και "Y" γίνονται:

```
X=100*COS(A)      Y=100*SIN(A)
```

Για τον κύκλο μας θα χρησιμοποιήσουμε μια τιμή ακτίνας 100.

Για να σχεδιάσουμε έναν κύκλο χρειαζόμαστε να σχεδιάσουμε κάθε σημείο πάνω στην περιφέρειά του και για να γίνει αυτό θα χρειαστεί μια FOR...NEXT ανακύκλωση. Θα ανακυκλωθεί από το 0 ως 2*PI ακτίνια ή αν προτιμάτε από 0 ως 360 μοίρες (ένας πλήρης κύκλος). Το μέγεθος του βήματος που χρησιμοποιείται επιτρέπει στον υπολογιστή να σχεδιάσει έναν πλήρη κύκλο, ένα μεγαλύτερο μέγεθος βήματος θα παρουσιάσει ένα διακεκομένο κύκλο.

ΠΡΟΓΡΑΜΜΑ 3.8

```
10 MODE 1
20 R=100
30 FOR A=0 TO 2*PI STEP 0.01
40 X=R*COS(A):Y=R*SIN(A)
50 PLOT 320+X,200+Y
60 NEXT A
```

Πως είναι αυτό; Ένας υπέροχος κύκλος δημιουργημένος με SIN και COS. Υπάρχει ωστόσο ένα πρόβλημα: ο κύκλος *παίρνει πολύ χρόνο να σχεδιαστεί*, περίπου 20 δευτερόλεπτα. Μη δυσφορείτε, είναι δυνατό να παρουσιαστεί ένας κύκλος σε πολύ λιγότερο χρόνο. Αυτό μπορεί να γίνει μ' έναν από τις δύο μεθόδους.

Πρώτα, μεγαλώστε το μέγεθος του ΒΗΜΑΤΟΣ. Πρέπει να δοθεί προσοχή όταν το κάνετε αυτό, γιατί η πολύ μεγάλη τιμή του βήματος θα παρουσιάσει ένα διακεκομένο κύκλο. Ένας πολύ λογικός κύκλος μπορεί να σχεδιαστεί χρησιμοποιώντας μια τιμή βήματος 0.02. Αυτό θα κόψει το

χρόνο σχεδίασης του κύκλου στο μισό, αλλά και αυτό είναι ακόμα πολύ αργό.

Η δεύτερη μέθοδος είναι να αντικαταστήσουμε την PLOT με DRAW. Αυτό κάνει το πρόγραμμα να σχεδιάζει μια γραμμή σε κάθε νέα συντεταγμένη. Επειδή η DRAW παρουσιάζει μια συμπληρωμένη γραμμή δεν θα υπάρχουν κενά στο κύκλο. Αυτό μας επιτρέπει να αυξήσουμε την τιμή σημαντικά. Ακόμα μια φορά, αν το μέγεθος του βήματος αυξηθεί πάρα πολύ το πρόβλημα θα ξαναπροκύψει και ο κύκλος θα εμφανιστεί περισσότερο σαν πολύγωνο. Ένα κατάλληλο μέγεθος ΒΗΜΑΤΟΣ είναι αυτό του 0.1, 10 φορές το μέγεθος που χρησιμοποιήθηκε στη γραμμή 30 του προγράμματος 3.8. Οι τροποποιήσεις ενσωματώθηκαν στο πρόγραμμα 3.8(a). Προσέξτε τη γραμμή 25, αυτή μετακινεί το δρομέα των γραφικών στο σημείο αφετηρίας του κύκλου.

ΠΡΟΓΡΑΜΜΑ 3.8(a)

```
10 MODE 1
20 R=100
25 PLOT 420,200
30 FOR A=0 TO 2*PI STEP 0.1
40 X=R*COS(A):Y=R*SIN(A)
50 DRAW 320+X,200+Y
60 NEXT A
```

Αφού πληκτρολογήσατε και τρέξατε το πρόγραμμα 3.8(a) θα παρατηρήσατε κάποιο μικρό πρόβλημα. Ο υπολογιστής δεν σχεδίασε ένα πλήρη κύκλο. Πιθανόν προς στιγμή να νομίσετε ότι ο υπολογιστής έκανε κάποιο λάθος, αλλά ΟΧΙ οι υπολογιστές κάνουν ότι ακριβώς τους λέτε, ΟΧΙ ότι νομίζετε ότι τους λέτε.

Για να κατανοήσετε το πρόβλημα πρέπει να ανατρέξετε σε προηγούμενα κεφάλαια με τις εντολές FOR-NEXT. Στο κεφάλαιο 2 μάθαμε ότι ο αθροιστής αυξάνεται έως ότου γίνει μεγαλύτερος ή ίσος του ορίου. Στην περίπτωση του προγράμματος 3.8(a) ο μετρητής Α παίρνει τις τιμές 5.9, 6.0, 6.1 και 6.2. Στην συνέχεια αυξάνεται σε 6.3 αλλά καθ όσον η τιμή αυτή είναι μεγαλύτερη απο το όριο (2*PI = 6.28318530), η ανακύκλωση σταματά χωρίς να σχεδιασθεί η περαιτέρω γραμμή. Αυτό σημαίνει ότι μιά μικρή τιμή δεν λαμβάνεται υπ όψη. Η τιμή αυτή περίπου 0.08 ίση με το βήμα 0.1 προκαλεί αυτή την ασυνέχεια.

Γενικά για να αποφύγουμε το πρόβλημα πρέπει να είμαστε βέβαιοι ότι το βήμα αύξησης διαιρείται ακριβώς με την διαφορά μεταξύ αρχής και τέλους. Μια απλή λύση του προβλήματος είναι η προσέγγιση του αριθμού 2*PI στην

γραμμή 30 σε 6.3. Έτσι το βήμα θα διαιρείται ακριβώς με το 2.3 και θα σχεδιασθεί πλήρης κύκλος.

ΑΣΚΗΣΗ 3.5

Σχεδίασε ένα ημικύκλιο με τη χρήση των SIN και COS. Το κέντρο του θα είναι στη θέση 120, 200 και η ακτίνα του 57. Μια πιθανή απάντηση δίνεται στο κεφάλαιο των λύσεων.

ΚΕΦΑΛΑΙΟ 4

4.1 Αλφαριθμητικές μεταβλητές και η δομή τους: Το παιχνίδι του αναγραμματισμού

Στο κεφάλαιο 2 αναπτύχθηκε το παιχνίδι της μάντεψης ενός αριθμού το οποίο χρησιμοποιούσε έναν τυχαίο αριθμό που παραγόταν από τον Amstrad. Στο Κεφάλαιο αυτό, θα γραφτεί ένα παρόμοιο παιχνίδι, χρησιμοποιώντας όμως αυτή τη φορά λέξεις, ένα παιχνίδι αναγραμματισμού. Έτσι, αντί να ζητείται από τον παίκτη να μαντέψει έναν αριθμό, θα απαιτείται από αυτόν να μαντέψει μία λέξη. Πρώτ' απ' όλα όμως, αυτό το κεφάλαιο θα ερευνήσει τους τρόπους αποθήκευσης αυτών των λέξεων και μετά την εμφάνισή τους, μία προς μία, όταν αυτό απαιτείται. Αυτό, θα πραγματοποιηθεί χρησιμοποιώντας τεχνικές δομημένου προγραμματισμού, για να παρουσιάσουμε πως ένα πολύπλοκο πρόβλημα μπορεί να διαιρεθεί σε εύκολα αντιληπτές υποδιαιρέσεις.

Κάθε ξεχωριστό μέρος του παιχνιδιού αναγραμματισμού, θ' αναπτυχθεί σαν μία ενότητα. Η ενότητα είναι ένα τμήμα του προγράμματος που εκτελεί μια σημαντική διαδικασία. Για παράδειγμα το παιχνίδι αναγραμματισμού, που αναπτύσσεται σ' αυτό το κεφάλαιο, έχει μια ενότητα για να διαλέγει μια τυχαία λέξη και μια άλλη ενότητα για ν' ανακατεύει τα γράμματα.

Όταν έχει να κάνει με τυχαίους αριθμούς, ο Amstrad μπορεί να παράγει ένα ατέλειωτο απόθεμα από αυτούς, αν του ζητηθεί. Μέσα στη ROM του (Read Only Memory) έχει ένα πρόγραμμα που παράγει τυχαίους αριθμούς, τόσο γρήγορα, όσο μπορούν να χρησιμοποιηθούν. Βέβαια, όταν έχεις να κάνεις με λέξεις, είναι αδύνατο να γίνει το ίδιο πράγμα. Οι υπολογιστές δεν ξέρουν τίποτε σχετικά με τις λέξεις, και έτσι όλες οι λέξεις που πρόκειται να χρησιμοποιηθούν θα πρέπει ν' αποθηκευθούν κάπου μέσα στο πρόγραμμα, και να προσδιοριστούν από τον προγραμματιστή. Ένας κοινός τρόπος αποθήκευσης τέτοιων δεδομένων, είναι σε αλφαριθμητικές μεταβλητές και το πρόγραμμα θα μπορούσε να περιέχει τέτοιες εντολές όπως:

```
LET A$="AMSTRAD"
LET B$="KEYBOARD"
LET C$="SCREEN"
```

4.1.1 READ...DATA

Αυτός ωστόσο ο τρόπος είναι εξαιρετικά κουραστικός, για να κάνετε τη δουλειά αυτή και γι' αυτό η BASIC εξασφαλίζει μία εναλλακτική μέθοδο, χρησιμοποιώντας δύο εντολές τη READ και τη DATA. Η πρώτη απ' αυτές λέει στον υπολογιστή να διαβάσει (READ) ένα κομμάτι δεδομένων και η δεύτερη του λέει που να βρει το δεδομένο. Η εντολή DATA είναι ένα κομμάτι προγράμματος που δεν έχει συγκεκριμένη θέση, μπορεί να μπει οπουδήποτε μέσα στο πρόγραμμα, συνηθίζεται όμως να μπαίνει ακριβώς στο τέλος και να μη μπαίνει στη μέση του προγράμματος. Το πρόγραμμα 4.1 διαβάζει, με τη γραμμή 100 ένα δεδομένο από τη DATA το οποίο εκτυπώνεται με τη γραμμή 110, ενώ η DATA έχει από την αρχή εισαχθεί στη γραμμή 9000.

ΠΡΟΓΡΑΜΜΑ 4.1

```
100 READ A$
110 PRINT A$
9000 DATA ONE, TWO, THREE
```

'Οταν τρέξει αυτό το πρόγραμμα θα πάρει μόνο ένα κομμάτι της DATA δηλαδή ONE και θα το εμφανίσει πάνω στην οθόνη. Αλφαριθμητικά δεδομένα δε χρειάζεται να περικλείονται από εισαγωγικά σε μια εντολή DATA. Αν το αλφαριθμητικό δεδομένο περιέχει ένα κόμμα όπως στο JONES,ED, τότε πρέπει να το κλείσετε σε εισαγωγικά. Για παράδειγμα, δοκιμάστε το 9000 DATA "JONES,ED",ONE,TWO στο πρόγραμμα 4.1.

Το πρόγραμμα 4.2 είναι ένα *παρόμοιο πρόγραμμα* όπου αντί για λέξεις αποθηκεύονται αριθμοί.

ΠΡΟΓΡΑΜΜΑ 4.2

```
100 READ A
110 PRINT A
9000 DATA 1,2,3
```

Οι διαφορές που έχει το πρόγραμμα αυτό σε σχέση με το πρόγραμμα 4.1 είναι πράγματι αυτές που θα περίμενε κανείς: Η αριθμητική μεταβλητή A αντικαθιστά την A$. Σε καμμία περίπτωση δεν περικλείονται σε εισαγωγικά αριθμητικά δεδομένα.

Οι εντολές READ μπορεί να είναι απλές ή πολύ περίπλοκες όσο ένα πρόγραμμα απαιτεί και ένας αριθμός μεταβλητών, θα μπορούσε να διαβαστεί από μία μόνο γραμμή προγράμματος: π.χ. READ A$,A,B$. Παρ' όλα αυτά όταν διαβάζουμε δεδομένα θα πρέπει να δώσουμε μεγάλη προσοχή για να είμαστε σίγουροι ότι η εντολή READ όταν προσπαθεί να διαβάσει ένα αριθμό, να βρίσκει αριθμητικό και όχι ένα αλφαριθμητικό δεδομένο. Αν δεν ταιριάζουν οι τύποι των μεταβλητών, ο υπολογιστής αναφέρει "Syntax error" (συντακτικό λάθος) στη γραμμή των δεδομένων όπου βρέθηκε απρόσμενα αλφαριθμητικό δεδομένο. Μια που η εντολή READ είναι σχετικά απλή δοκιμάστε αυτή τη μικρή άσκηση.

ΑΣΚΗΣΗ 4.1

Γράψε ένα πρόγραμμα που θα διαβάζει τους αριθμούς από 1 μέχρι 4 από εντολές DATA ταυτόχρονα με γράμματα και αριθμούς και να τα εμφανίσει στην οθόνη ως εξής:

1 ONE
2 TWO
3 THREE
4 FOUR

Μία πιθανή λύση δίνεται στο κεφάλαιο λύσεων.

'Εχοντας βρει έναν τρόπο αποθήκευσης και επαναφοράς δεδομένων πρέπει να βρεθεί κάποιος τρόπος ώστε να μπορούμε να επιλέξουμε τυχαία ένα από αυτά χρησιμοποιώντας ένα βρόχο FOR...NEXT για να διαβάσει ένα ορισμένο πλήθος στοιχείων από την DATA. 'Ενα συγκεκριμένο κομμάτι μπορεί να ανακληθεί όπως φαίνεται στο πρόγραμμα 4.3. Σ' αυτό το πρόγραμμα ο βρόχος εκτελείται τρεις φορές και έτσι ανακαλείται το τρίτο κομμάτι από τα δεδομένα.

ΠΡΟΓΡΑΜΜΑ 4.3

```
120 FOR X=1 TO 3
130 READ A$
140 NEXT X
150 PRINT A$
9000 DATA ONE,TWO,THREE
```

Στην πραγματικότητα τρία μέρη δεδομένων έχουν ανακληθεί αλλά μόνο η αλφαριθμητική THREE τυπώνεται. Στο πρώτο πέρασμα του βρόχου, η τιμή του A$ θα έπρεπε να είναι ONE, στο δεύτερο πέρασμα, θ' αντικατασταθεί με το TWO και τελικά με το THREE. Αυτή ήταν η αλφαριθμητική THREE που αποθηκεύτηκε στην A$ τη στιγμή που η γραμμή 150 εμφάνισε την αλφαριθμητική μεταβλητή πάνω στην οθόνη.

Σε κάθε πέρασμα του βρόχου, η γραμμή 130 διαβάζει το επόμενο στοιχείο της εντολής DATA. Ξέρει βέβαια, ποιο στοιχείο είναι επόμενο, αφού κάθε φορά που εκτελείται μια εντολή READ, ένας δείκτης μετακινείται διαδοχικά από το ένα δεδομένο στο άλλο που θα διαβαστεί. Αυτό, μπορεί να *προκαλέσει προβλήματα*, αν γίνει προσπάθεια να διαβαστούν περισσότερα δεδομένα απ' όσα υπάρχουν. Για παράδειγμα αν το πρόγραμμα 4.3 μετατραπεί έτσι ώστε να διαβάζει περισσότερο από μιά φορά τα δεδομένα, τότε δεν θα υπάρχουν αρκετά (δεδομένα) και ο υπολογιστής θα το αναφέρει. Δοκιμασέ το, προσθέτοντας το πρόγραμμα 4.3(a) στο πρόγραμμα 4.3.

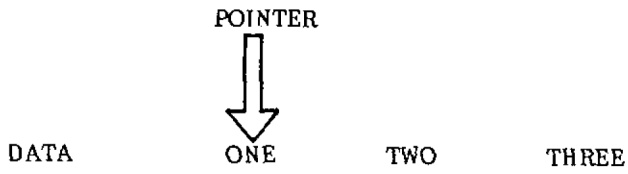
ΠΡΟΓΡΑΜΜΑ 4.3(a)

```
160 GOTO 120
```

Όταν εκτελεστεί το πρόγραμμα 4.3(a) ο Υπολογιστής θ' αναφέρει τα παρακάτω:

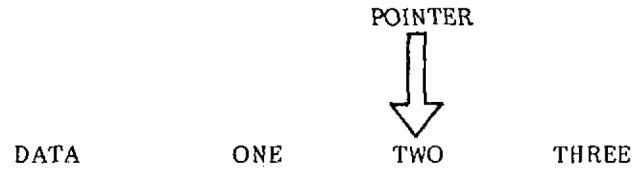
```
DATA exhausted in 130
(Εξάντληση δεδομένων στη γραμμή 130)
```

Το οποίο απλά σημαίνει ότι έκανε READ όλα τα DATA (δεδομένα) και ότι ψάχνει για άλλα αλλά δεν τα βρίσκει. Για να παρακολουθήσει που βρίσκεται μέσα στα δεδομένα η BASIC της Amstrad χρησιμοποιεί ένα δείκτη και αυτός μετακινείται για να δείξει το επόμενο σημείο του DATA που θα γίνει READ. Για παράδειγμα πριν διαβαστεί η γραμμή 130 του προγράμματος 4.3(a), η δομή της εντολής DATA του Amstrad είναι όπως στο σχήμα 4.1.



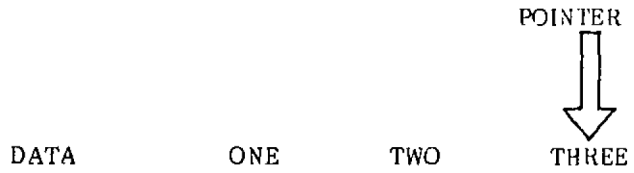
Εικόνα 4.1

Έτσι, στο πρώτο "READ A$", η μεταβλητή A$ παίρνει την τιμή "ONE" και ο δείκτης μετακινείται στο δεδομένο "TWO" όπως στο σχήμα 4.2.



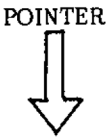
Εικόνα 4.2

Την επόμενη φορά η μεταβλητή A$ παίρνει την τιμή "TWO" και ο δείκτης μετακινήθηκε ξανά. Σχήμα 4.3.



Εικόνα 4.3

Την τελευταία φορά που εκτελείται ο βρόχος η μεταβλητή A$ παίρνει την τιμή "THREE" και για μια φορά ακόμη, ο δείκτης μετατοπίζεται σε μια θέση μετά το "THREE", δείχνοντας το κενό.



Εικόνα 4.4

Στο σημείο αυτό το πρόγραμμα περνά το NEXT X και η γραμμή 150 τυπώνει την τιμή της μεταβλητής A$ που είναι τη στιγμή αυτή, THREE. Μετά απ' αυτό η γραμμή 160 ξανακατευθύνει το πρόγραμμα στη γραμμή 120 και ο βρόχος FOR...NEXT ξαναρχίζει.

'Οταν συναντηθεί η γραμμή 130 ξανά, ο Υπολογιστής βλέπει ότι ο δείκτης δείχνει το κενό, δείχνοντας έτσι ότι δεν υπάρχουν άλλα δεδομένα. Στο στάδιο αυτό ο Υπολογιστής δίνει το μήνυμα:

DATA exhausted in 130

Σε πολλές περιπτώσεις ένα πρόγραμμα χρειάζεται να χρησιμοποιεί τα ίδια DATA ξανά και για να γίνει αυτό, πρέπει ο δείκτης να μετακινηθεί στην αρχή της DATA. Αυτό πραγματοποιείται μέσω της εντολής της BASIC:

4.1.2 RESTORE

Αυτή έχει το αποτέλεσμα να μετακινήσει πίσω το δείκτη, στην αρχή της DATA. Αυτό φαίνεται στο πρόγραμμα 4.4 όπου η εντολή RESTORE εκτελείται πριν να σταλεί το πρόγραμμα πίσω, για να ξαναρχίσει το βρόχο.

ΠΡΟΓΡΑΜΜΑ 4.4

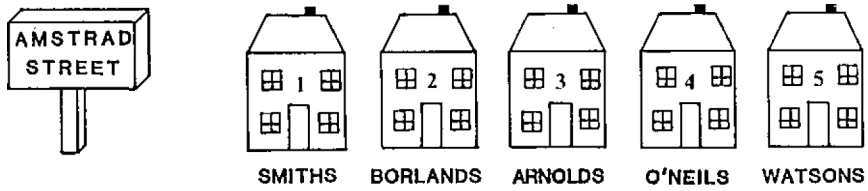
```
120 FOR X=1 TO 3
130 READ A$
140 NEXT X
150 PRINT A$
160 RESTOR:GOTO 120
9000 DATA ONE, TWO, THREE
```

'Οταν το πρόγραμμα 4.4 εκτελείται, τρέχει μέσα από τα δεδομένα (DATA) ατελείωτα. Ωστόσο, κάθε φορά που περνά από το βρόχο και φτάνει στη μεταβλητή χαρακτήρων THREE, αυτό καθορίζεται από το βρόχο για τιμή 3. Για το

παιχνίδι αναγραμματισμού η λέξη που πρέπει να μαντέψει κανείς, πρέπει να επιλεγεί τυχαία. Αυτό μπορεί να γίνει χρησιμοποιώντας μια ειδική μορφή μεταβλητής που ονομάζεται πίνακας (Array).

4.2 Πίνακες

Ένας πίνακας είναι μια σειρά, ένας κατάλογος από σχετικές μεταβλητές. Είναι συσχετισμένες στ' ότι έχουν το ίδιο όνομα. Ένας δρόμος μπορεί να φαίνεται κατά κάποιο τρόπο σαν πίνακας. Ο δρόμος έχει ένα όνομα και όλα τα σπίτια έχουν έναν αριθμό, σχήμα 4.5.



Εικόνα 4.5

Όπως δείχνει το διάγραμμα, οι Arnolds ζουν στον αριθμό τρία της οδού Amstrad. Αυτό, θα μπορούσε να γραφεί κάπως έτσι:

```
AMSTRAD STREET(3)=ARNOLDS
```

όπου "Amstrad Street(3)" είναι τ' όνομα του πίνακα, "3" ο δείκτης του πίνακα και Arnolds η τιμή του. Αν όλες οι λέξεις στο παιχνίδι μας ήταν ν' αποθηκευτούν κάτω από το συλλογικό όνομα WORD$, τότε η πρώτη λέξη θα ήταν η WORD$(1), η δεύτερη η WORD$(2) κ.λ.π. Αυτό αποδεικνύεται στο πρόγραμμα 4.4.

ΠΡΟΓΡΑΜΜΑ 4.4(a)

```
10 FOR X=1 TO 4
20 READ WORD$(X)
30 NEXT X
9000 DATA FLOWER,RAIN,AMSTRAD,COMPUTER
```

Αφού τρέξει το πρόγραμμα, ο πίνακας WORD$ θα περιέχει τις παρακάτω τιμές:

```
WORD$(1)="FLOWER"                    WORD$(2)="RAIN"
WORD$(3)="AMSTRAD"                   WORD$(4)="COMPUTER"
```

Ο τυχαίος αριθμός μπορεί τώρα, να επιλέξει μέσα από τον πίνακα. Το πρόγραμμα 4.4(b) το δείχνει:

ΠΡΟΓΡΑΜΜΑ 4.4(b)

```
40 R=INT(RND*4)+1
50 A$=WORD$(R)
60 PRINT A$
```

Η γραμμή 40 παράγει έναν τυχαίο αριθμό, που μετά αποθηκεύεται στη μεταβλητή R. Μετά, η R τιμή του πίνακα WORD$ αποθηκεύεται στην A$ και κατόπιν τυπώνεται στην οθόνη. Όταν τρέξαμε το πρόγραμμα, εμφανίστηκε στην οθόνη η FLOWER. Χρησιμοποιώντας αυτή τη μέθοδο, όλες οι λέξεις για το παιχνίδι αναγραμματισμού μπορούν να αποθηκευτούν στο πίνακα WORD$. Το πρόγραμμα 4.4(c) χρησιμοποιεί τη WORD$ για να αποθηκεύσει είκοσι διαφορετικές λέξεις.

ΠΡΟΓΡΑΜΜΑ 4.4(c)

```
10 FOR X=1 TO 20
20 READ WORD$(X)
30 NEXT X
40 R=INT(RND*20)+1
50 A$=WORD$(R)
60 PRINT A$

9000 DATA FLOWER, RAIN, AMSTRAD, COMPUTER
9010 DATA ADAPT, CREATE, IMAGINE, FRUIT, WALL
9020 DATA CONFUSION, STRANGE, BEAUTIFUL
9030 DATA PLASTIC, ELASTIC, BOMBASTIC, GRAND
9040 DATA YESTERDAY, NEWSPAPER, POT, PEANUT
```

Αυτό που θα έπρεπε να κάνει το πρόγραμμα είναι να διαβάσει είκοσι λέξεις μέσα στον πίνακα WORD$ και μετά τυχαία να τυπώσει στην οθόνη μία από τις λέξεις. Αυτό που πραγματικά συμβαίνει, είναι άλλη υπόθεση. Το πρόγραμμα δίνει το μήνυμα:

```
Subscript out of range in 20
```

Αυτός είναι ο τρόπος του Amstrad για να σας πει ότι

προσπαθείτε να βάλετε πάρα πολλές τιμές σ' έναν πίνακα. Αν δεν έχετε πει προηγουμένως στον υπολογιστή πόσες τιμές μπορεί να κρατήσει ο πίνακας WORD$, τότε υποθέτει ότι αυτές είναι δέκα. Κάθε προσπάθεια να διαβάσετε ή να εκτυπώσετε την 11η τιμή του WORD$ θάχει σαν αποτέλεσμα, ένα μήνυμα που θα σας δίνει λάθος έξω από την δυνατότητα κλίματος.

Η εντολή που λέει στον υπολογιστή πόσες τιμές ενός πίνακα χρειαζόμαστε, λέγεται:

4.2.1 DIM

Η εντολή DIM (DIM είναι μια συντομογραφία του DIMENSION) λέει στον υπολογιστή να τοποθετήσει ένα συγκεκριμένο αριθμό θέσεων για μια συγκεκριμένη μεταβλητή. Π.χ.

```
DIM WORD$(20)
```

Θα φυλάξει είκοσι θέσεις για τον πίνακα WORD$. Τώρα αν προσθέσουμε το πρόγραμμα 4.4(c) στο πρόγραμμα 4.4(d) θα δουλέψει σωστά.

ΠΡΟΓΡΑΜΜΑ 4.4(d)

```
5 DIM WORD$(20)
```

Μια ιδιορυθμία του DIM είναι ότι δεν μπορείτε να δώσετε διάσταση στον ίδιο πίνακα, περισσότερο από μια φορά σ' ένα πρόγραμμα. Για παράδειγμα αν η γραμμή 7 του προγράμματος 4.4(c) προσθετόταν και εκτελεστεί όλο το πρόγραμμα μαζί, τότε θα εκτυπωθεί το ακόλουθο μήνυμα λάθους:

ΠΡΟΓΡΑΜΜΑ 4.4(e)

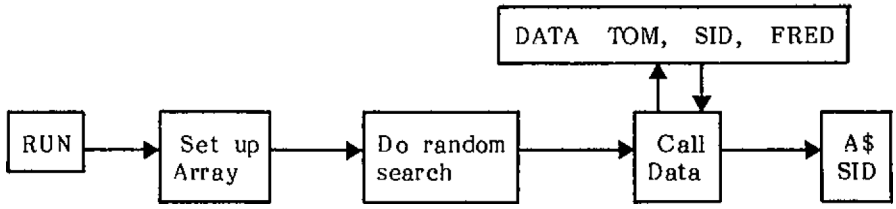
```
7 DIM WORD$(20)

    Array already dimensioned in 7
```

Αυτό το μήνυμα λάθους λέει στο χρήστη, καθαρά ότι η διάσταση του πίνακα που ορίζεται στη γραμμή 7, έχει γίνει ήδη DIM προηγούμενα στο πρόγραμμα. Οι πίνακες δουλεύουν με τον ίδιο τρόπο με τις αριθμητικές μεταβλητές. Σαν απόδειξη είναι το παρακάτω μικρό πρόγραμμα.

ΠΡΟΓΡΑΜΜΑ 4.4(f)

```
1 DIM T(50)
2 FOR X=1 TO 50:T(X)=X:NEXT X
3 FOR X=1 TO 50
4 PRINT T(X);:NEXT:STOP
```

Στο τρέξιμο του προγράμματος ο υπολογιστής θα εμφανίσει τους αριθμούς που είναι τα περιεχόμενα των μεταβλητών T(1) ως T(50). Πριν συνεχίσουμε το παιχνίδι του αναγραμματισμού, διαγράψτε τις γραμμές 1-4.



Module 1

Εικόνα 4.6

Το σχήμα 4.6 παριστάνει διαγραμματικά πως το ιδιαίτερο τμήμα λειτουργεί. Εκτελέστε το και θα βρείτε ένα τυχαία επιλεγμένο κομμάτι της DATA στην A$.

Καθώς θα χρησιμοποιήσουμε αργότερα αυτό το κομμάτι του προγράμματος ας το αποθηκεύσουμε ξεχωριστά αριθμώντας τις εντολές του από τον αριθμό γραμμής 1000 και πάνω.

4.2.2 RENUM

Η εντολή RENUM της Amstrad BASIC χρησιμοποιείται για να αριθμήσει τις γραμμές. Η RENUM έχει τρία ορίσματα το πρώτο είναι από που θέλετε ν' αρχίζει ο αριθμός της καινούργιας γραμμής εντολής. Στο παράδειγμα αυτό θα είναι η γραμμή 1000. Η δεύτερη τιμή, είναι ο αριθμός της γραμμής που θέλετε ν' αλλαχτεί, στο παράδειγμα η γραμμή 120. Η τρίτη και τελευταία παράμετρος, είναι το μέγεθος του βήματος που θέλεις ν' αυξάνονται οι αριθμοί των γραμμών εντολών. Θα χρησιμοποιήσουμε το βήμα 10.

	Νέοι αριθμοί θα αρχίσουν στη γραμμή	,	γραμμή από την οποία θα ξεκινήσει η επαναρίθμηση	,	βήμα αύξησης της αρίθμησης γραμμών
RENUM					

Στο παράδειγμά μας η εντολή RENUM θα είναι ως εξής:

```
RENUM 1000,5,10
```

Όλες οι τιμές της εντολής RENUM είναι επιλεγόμενες. Αν μείνουν έξω θ' αντικατασταθούν από την τιμή 10. Για παράδειγμα, πληκτρολογώντας RENUM χωρίς παραμέτρους θα προκαλέσει την αρίθμηση του προγράμματος, αρχίζοντας από τη γραμμή 10 και αυξάνοντας με βήματα των 10. Πριν κάνετε αυτό, διαγράψτε τη γραμμή 60.

ΠΡΟΓΡΑΜΜΑ 4.5

```
1000 DIM WORD$(20)
1010 FOR X=1 TO 20
1020 READ WORD$(X)
1030 NEXT X
1040 R=INT(RND*20)+1
1050 A$=WORD$(R):AA$=A$
1060 DATA FLOWER, RAIN, AMSTRAD, COMPUTER
1070 DATA ADAPT, CREATE, IMAGINE, FRUIT, WALL
1080 DATA CONFUSION, STRANGE, BEAUTIFUL
1090 DATA PLASTIC, ELASTIC, BOMBASTIC, GRAND
1100 DATA YESTERDAY, NEWSPAPER, POT, PEANUT
```

Αυτό το συγκεκριμένο τμήμα του προγράμματος χρειάζεται να χωριστεί σε δύο μέρη. Το πρώτο μέρος που οργανώνει τον πίνακα θα κληθεί μόνο μία φορά στο ξεκίνημα του προγράμματος. Το δεύτερο μέρος είναι αυτό που τυχαία διαλέγει τη λέξη. Μια άλλη εντολή RENUM μπορεί να χρησιμοποιηθεί για να χωρίσει τα δύο τμήματα. Πληκτρολογείστε:

```
RENUM 1100,1040,10
```

Το πρόγραμμα τώρα θα δείχνει κάπως έτσι:

```
1000 DIM WORD$(20)
1010 FOR X=1 TO 20
1020 READ WORD$(20)
1030 NEXT X

1100 R=INT(RND*20)+1
1110 A$=WORD$(R):AA$=A$
1120 DATA FLOWER, RAIN, AMSTRAD, COMPUTER
1130 DATA ADAPT, CREATE, IMAGINE, FRUIT, WELL
1140 DATA CONFUSION, STRANGE, BEAUTIFUL
1150 DATA PLASTIC, ELASTIC, BOMBASTIC, GRAND
1160 DATA YESTERDAY, NEWSPAPER, POT, PEANUT
```

Η γραμμή 1110 έχει αλλαχθεί για να δημιουργήσει ένα δεύτερο αντίγραφο της τυχαίας επιλεγόμενης λέξης. Το πρώτο αντίγραφο, αποθηκευμένο στην αλφαριθμητική μεταβλητή A$, θα χρησιμοποιηθεί για να δημιουργήσει την κόπια του αναγράμματος.

4.3 Τεμαχίζοντας λέξεις

Στο παιγνίδι του αναγραμματισμού που αναπτύσσουμε είναι αναγκαίο να τεμαχίσουμε τις λέξεις για να ξεχωρίσουμε τα γράμματά τους. Για να γίνει αυτό χρησιμοποιούμε τις συναρτήσεις της BASIC:

4.3.1 LEFT$, RIGHT$ ΚΑΙ MID$

Η BASIC της Amstrad εξασφαλίζει αρκετούς τρόπους για να κομματιάζει κανείς αλφαριθμητικές μεταβλητές. Δύο απ' αυτές είναι η LEFT$ και η RIGHT$. Αυτές, απλά κόβουν τις αριστερές και δεξιές άκρες των αλφαριθμητικών μεταβλητών αντίστοιχα. Ας προσπαθήσουμε να κομματιάσουμε μερικές τέτοιες αλφαριθμητικές μεταβλητές για εξάσκηση. Πρώτα ας θέσουμε A$=COMPUTER". Για να χρησιμοποιήσουμε την έννοια θα "θέσουμε την τιμή της μεταβλητής A$ στην κυριολεκτική τιμή COMPUTER. Η πρώτη εξάσκηση θάναι με LEFT$. Η LEFT$ δίνει το συγκεκριμένο αριθμό αριστερών χαρακτήρων μιας αλφαριθμητικής μεταβλητής και παίρνει τη μορφή:

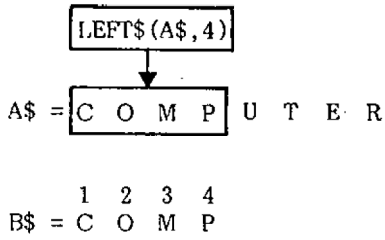
LEFT$(X$,N)

όπου X$ είναι η αλφαριθμητική μεταβλητή (string) που θέλουμε να κομματιάσουμε και N είναι αριθμός των χαρακτήρων που θέλουμε να μετακινήσουμε.

Στο παράδειγμά μας η αλφαριθμητική μεταβλητή χαρακτήρων είναι η A$ που της δόθηκε η τιμή COMPUTER. Αν

θα θέλαμε να μετακινήσουμε τους 4 πρώτους χαρακτήρες π.χ. COMP θα χρησιμοποιήσουμε την παρακάτω μέθοδο:

```
A$="COMPUTER"
B$=LEFT$(A$,4)
PRINT B$
```



Εικόνα 4.7

Η εντολή RIGHT$() δουλεύει ακριβώς με τον ίδιο τρόπο, με τη διαφορά ότι αρχίζει να μετρά από τη δεξιά πλευρά της αλφαριθμητικής μεταβλητής. Η RIGHT$() παίρνει την παρακάτω μορφή:

```
RIGHT$(X$,N)
```

όπου X$ είναι αλφαριθμητική μεταβλητή και N είναι ο αριθμός των χαρακτήρων που θα μετακινηθούν.

Έτσι αν A$="COMPUTER", C$=RIGHT$ (A$,5) θα τοποθετήσει στη C$ την τιμή "PUTER" τους πέντε πιο δεξιούς χαρακτήρες του "COMPUTER". Δοκιμάστε το, για να βεβαιωθείτε!

Αντίθετα η συνάρτηση MID$ μπορεί να ξεκινήσει οπουδήποτε μέσα στην αλφαριθμητική μεταβλητή. Επιτρέπει στον προγραμματιστή ν' αποχωρίσει επιλεκτικά λίγους ή πολλούς χαρακτήρες από τη μεταβλητή πάνω στην οποία εργάζεται. Η MID$ έχει τη μορφή:

```
MID$(X$,S,N)
```

όπου X$ είναι η αλφαριθμητική μεταβλητή που θα "κομματιαστεί", S είναι η θέση απ' όπου θα ξεκινήσει η ενέργεια και N είναι ο αριθμός των χαρακτήρων που θα μετακινηθούν. Μπορεί να χωρίσει είτε από το μέσον, είτε από ένα από τα δύο άκρα της αλφαριθμητικής μεταβλητής. Ας το εξετάσουμε λεπτομερέστερα με το παράδειγμα:

```
A$ = "COMPUTER"
C$ = MID$(A$,4,3)
```

* Όταν ο υπολογιστής δει C$=MID$(...), ξέρει ότι η αλφαριθμητική μεταβλητή θα κομματιαστεί και το αποτέλεσμα θα αποθηκευτεί στην αλφαριθμητική μεταβλητή C$.

* Έπειτα συνεχίζει και βλέπει MID$(A$...). Αυτό το μεταφράζει σε "πρώτα βρες την A$ και ετοιμάσου να εργαστείς σ' αυτήν".

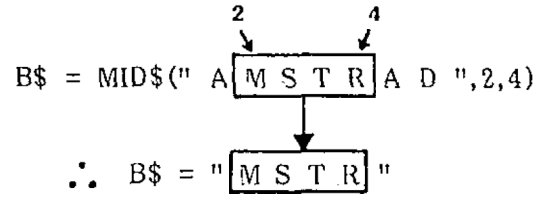
* Κατόπιν βλέπει το "4" στην MID$(A$,4...) και αυτό του λέει να ξεκινήσει από τον τέταρτο χαρακτήρα της μεταβλητής.

* Μετά, διαβάζει το "3" στην MID$(A$,4,3) και ξεκινώντας από τον τέταρτο χαρακτήρα της αλφάριθμητικής μεταβλητής, αφαιρεί τρεις χαρακτήρες. Αυτούς τους αποθηκεύει στην C$. Έτσι, ακολουθώντας την εργασία, η C$ θα περιέχει "PUT".

Σε γενική μορφή η δομή της εντολής αυτής είναι:

MID$(A$,START,LENGTH)

"START"και "LENGTH" πρέπει να είναι και οι δύο αριθμοί. Η MID$ θα αποκόψει μέρος της A$, ξεκινώντας από τον αριθμό χαρακτήρων "START" και μήκους χαρακτήρων "LENGTH". Διαγραμματικά η MID$ εμφανίζεται σαν:



Εικόνα 4.8

Χρησιμοποιώντας αυτές τις εντολές χειρισμού αλφαριθμητικών μεταβλητών σε καθιστά ικανό να κομματιάζεις μία λέξη που θέλεις, παίρνοντας ξεχωριστά γράμματα όπως ζητείται. Έτσι, κάθε φορά που ένα γράμμα ζητείται από οποιοδήποτε σημείο μιας λέξης μπορεί να ληφθεί με τη βοήθεια της DIM$. Το πρόγραμμα 4.6 δείχνει την ενέργεια αυτή, καθώς το MID$ διαβάζει βήμα με βήμα το εσωτερικό μιας λέξης, τυπώνοντας κάθε γράμμα καθώς προχωρεί.

```
1500 LET A$=COMPUTER"
1510 FOR X=1 TO 8
1520 B$=MID$(A$,X,1)
1530 PRINT B$
1540 NEXT X
```

Ένα από τα προβλήματα που θα μπορούσε να παρουσιάσει το πρόγραμμα 4.6, αν δοκιμάζαμε και χρησιμοποιούσαμε αυτό σαν τμήμα ενός προγράμματος, είναι ότι δουλεύει όταν η λέξη έχει οκτώ γράμματα, αλλά για λέξεις με περισσότερα ή λιγότερα γράμματα, θα αποδεικνυόταν να είναι προβληματικό. Αλλά, το έχετε μαντέψει η BASIC έχει μία λύση γι' αυτό με τη συνάρτηση:

4.3.2 LEN()

Αυτή η εντολή χρησιμοποιείται για να δηλώσει πόσοι χαρακτήρες υπάρχουν σε μία συγκεκριμένη αλφαριθμητική μεταβλητή δηλαδή, η LEN() επιστρέφει (δίνει) το μήκος της αλφαριθμητικής μεταβλητής. Ελέγξτε το με:

```
PRINT LEN("COMPUTER")
```

και μετά με μερικές άλλες αλφαριθμητικές μεταβλητές κάθε φορά πρέπει να τυπώνει το μήκος της αλφαριθμητικής μεταβλητής που χρησιμοποιεί. Το πρόγραμμα 4.6 μπορεί τώρα να ξαναγραφτεί έτσι ώστε ο βρόχος θα διατρέχει στο σωστό αριθμό φορών, οποιοδήποτε κι αν είναι το μήκος της μεταβλητής, όπως στο πρόγραμμα 4.7.

ΠΡΟΓΡΑΜΜΑ 4.7

```
1500 LET A$="COMPUTER"
1510 FOR X=1 TO LEN(A$)
1520 B$=MID$(A$,X,1)
1530 PRINT B$
1540 NEXT X
```

Οποιοδήποτε και αν είναι η αλφαριθμητική μεταβλητή που έχει εκχωρηθεί στο A$, η ανακύκλωση τώρα θα το χειρίζεται πάντα. Ωστόσο ο σκοπός αυτού του σχεδίου, είναι ένα παιχνίδι αναγραμματισμού έτσι η ενότητα που αναπτύσσουμε, θα μπορούσε να το ξανατακτοποιήσει και τα γράμματα όχι να τα αφήνει με την ίδια σειρά. Η γραμμή 1520, είναι αυτή που ανατέμνει την αλφαριθμητική μεταβλητή και το κάνει με μεθοδικό τρόπο, ξεκινώντας από 1 και προοδευτικά προχωρώντας μέσα από το LEN(A$). Ένας

τρόπος για να το κάνει με λιγότερο μεθοδικό τρόπο θα θα ήταν να αντικαταστήσει τη μεταβλητή X στη γραμμή 1520 μ' έναν τυχαίο αριθμό που βρίσκεται μεταξύ του 1 και του LEN(A$). Αυτό, μπορεί να γίνει θέτοντας μια μεταβλητή, έστω R, στον κατάλληλο τυχαίο αριθμό και αντικαθιστώντας τη γραμμή 1520 με:

```
B$=MID$(A$,R,1)
```

και ο τυχαίος αριθμός - θυμηθείτε την ασκηση του κεφαλαίου 2 όπου βρήκαμε, ότι χρειαζόταν να προσθέσουμε 1 για να παράγουμε τη σωστή κλίμακα.

```
R=INT(LEN(A$)*RND)+1
```

Αυτό μπορεί να προστεθεί στο πρόγραμμα 4.7:

ΠΡΟΓΡΑΜΜΑ 4.7

```
1515 R=INT(LEN(A$)*RND)+1
1520 B$=MID$(A$,R,1)
```

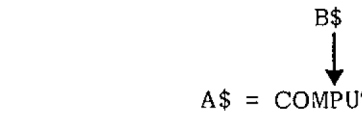
Έτσι όταν οι γραμμές 1515 και 1520 προσθέτονται στο πρόγραμμα 4.7 δίνει το παρακάτω πρόγραμμα 4.8:

ΠΡΟΓΡΑΜΜΑ 4.8

```
1500 LET A$="COMPUTER"
1510 FOR X=1 TO LEN(A$)
1515 R=INT(LEN(A$)*RND)+1
1520 B$=MID$(A$,R,1)
1530 PRINT B$
1540 NEXT X
```

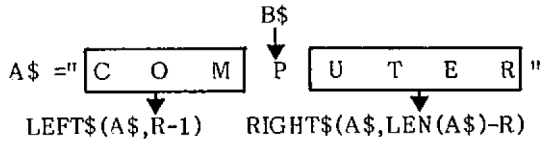
Τώρα, όταν αυτό τρέξει θα τυπώσει τα γράμματα της A$ με τυχαίο τρόπο. Υπάρχει όμως ένα πρόβλημα. Αν κοιτάξετε τα τυπωμένα γράμματα είναι σχεδόν βέβαιο ότι ένα ή περισσότερα έχουν επαναληφθεί. Κι αυτό γιατί, όταν ένα γράμμα έχει διαλεχθεί τυχαία μια φορά, παραμένει στην A$ για να διαλεχθεί ξανά! Αυτό που χρειάζεται να κάνουμε, είναι να μετακινούμε κάθε γράμμα που μαντέψαμε μια φορά, που είναι μια σαφώς δύσκολη δουλειά. Ωστόσο, μπορεί να γίνει χωρίς μεγάλη δυσκολία, χάρη στις εντολές LEFT$ και RIGHT$.

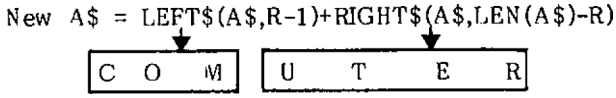
Έστω για παράδειγμα, ότι το τυχαίο επιλεγμένο γράμμα ήταν το "P" στη (λέξη) "COMPUTER" π.χ.



Για να "μετακινήσουμε" τη B$ θα πρέπει να πάρουμε το τμήμα της αλφαριθμητικής μεταβλητής αριστερά από το P και εκείνο, δεξιά στο P και να τα προσθέσουμε μαζί, ονομάζοντας αυτό καινούρια A$.

Καθώς η B$ είναι το γράμμα που καθορίζεται από την τυχαία τιμή R π.χ. το R-στο γράμμα, (στην περίπτωσή μας το τέταρτο) υπάρχουν R-1 γράμματα αριστερά απ' αυτό (στην περίπτωσή μας 3). Η αλφαριθμητική μεταβλητή στα δεξιά του B$ είναι ελαφρά πιο πολύπλοκη, καθώς συνίσταται απ όλη τη λέξη, εκτός των R χαρακτήρων δηλαδή LEN (A$)-R χαρακτήρες. Σ' αυτή την περίπτωση 8-4=4 χαρακτήρες. Η νέα λοιπόν τιμή της A$ αποτελείται από το κομμάτι στα αριστερά του R συν το κομμάτι από δεξιά, δηλαδή:





Εικόνα 4.9

Για να σχηματίσουμε τη νέα A$, πρέπει να προστεθούν μαζί τ' αριστερά και δεξιά μέρη σαν να ήταν αριθμοί. Όταν αυτό γίνει σε αλφαριθμητικές μεταβλητές του δίνεται η παράξενη ονομασία concatenation. Για να το δοκιμάσετε πληκτρολογήσετε την παρακάτω γραμμή:

```
A$="FRED":B$="DY":C$=A$+B$:PRINT C$
```

Αυτό θα τυπώσει "FREDDY".

Με τη χρήση αυτού του τεχνάσματος, το πρόγραμμα 4.8 μπορεί να μετατραπεί έτσι, ώστε να βγάζει έξω ένα γράμμα κάθε φορά, να κλείνει το υπολοίπο της A$ και μετά να προσθέτει το μετακινημένο γράμμα στη καινούρια αλφαριθμητική μεταβλητή. Το αποτέλεσμα αυτής της

πρόσθεσης - θ' αποθηκευτεί σε μια αλφαριθμητική μεταβλητή που καλείται AN$ (ANswer = απάντηση) που προοδευτικά θ' αυξάνεται κατά ένα γράμμα κάθε φορά, μέχρι να περιλάβει όλα τα γράμματα. Ωστόσο, για να ξεκινήσετε, η αλφαριθμητική μεταβλητή πρέπει ν' αδειάσει ή να τοποθετηθεί σε μια άδεια αλφαριθμητική μεταβλητή δηλαδή AN$="". Αν δεν είστε τόσο σίγουροι γι' αυτήν τη διαδικασία, εκτελέστε την παρακάτω, μικρή άσκηση. Πληκτρολογήστε το πρόγραμμα 4.9.

ΠΡΟΓΡΑΜΜΑ 4.9

```
1 A$="FRED":AN$="AND"
2 PRINT A$;AN$;A$
3 STOP
```

'Οταν εκτελέσετε αυτό το πρόγραμμα θα πάρετε την εμφάνιση.

FREDANDFRED

Τώρα τροποποιήστε το πρόγραμμα 4.9 όπως φαίνεται κάτω στο πρόγραμμα 4.9(a) με σκοπό να τοποθετήσετε στην AN$ μια άδεια αλφαριθμητική μεταβλητή (γνωστή σαν κενή αλφαριθμητική).

ΠΡΟΓΡΑΜΜΑ 4.9(a)

```
1 A$="FRED":AN$=""
```

'Οταν εκτελεστεί αυτό, η εμφάνιση θα πρέπει να δείχνει:

FREDFRED

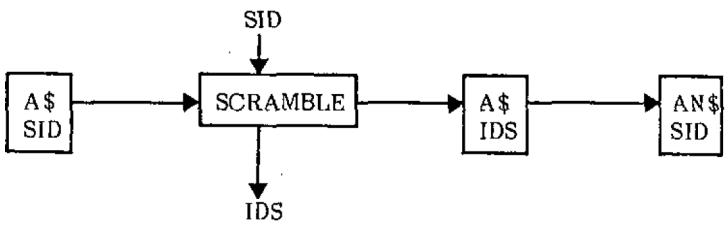
Μ' άλλα λόγια η αλφαριθμητική μεταβλητή AN$ είναι τώρα άδεια. 'Εχοντας δει όλα αυτά, διαγράψτε τις γραμμές 1-3. Εφαρμόζοντας αυτή την ιδέα στο πρόγραμμα 4.8 προκύπτει το πρόγρμμα 4.10.

ΠΡΟΓΡΑΜΜΑ 4.10

```
1500 LET A$="COMPUTER":AN$=""
1510 FOR X=1 TO LEN(A$)
1520 R=INT(LEN(A$)*RND)+1
1530 B$=MID$(A$,R,1)
1540 AN$=AN$+B$:PRINT AN$
1550 A$=LEFT$(A$,R-1)+RIGHT$(A$,LEN(A$)-R)
1560 NEXT X
```

Όταν τρέξει το πρόγραμμα 4.10 θα εκτυπώσει τον αναγραμματισμό στάδιο προς στάδιο, όπως θα δημιουργείται. Ακόμα καθώς η A$ μειώνεται προοδευτικά ο τυχαίος αριθμός που διαλέγει το γράμμα (R) μειώνεται επίσης προοδευτικά. Πάρα πολύ βολικό, ε;

Θέτοντας το σε ένα διάγραμμα και ονομάζοντας το ενότητα 2 δίνει το σχήμα 4.10.



Module 2

Εικόνα 4.10 Ενότητα 2.

Το πρόγραμμα 4.11 δείχνει την ενότητα με το ενδιάμεσο "PRINTS" να έχει μετακινηθεί.

ΠΡΟΓΡΑΜΜΑ 4.11 – Ενότητα 2

```
1500 AN$=""
1510 FOR X=1 TO LEN(A$)
1520 R=INT(LEN(A$)*RND)+1
1530 B$=MID$(A$,R,1)
1540 AN$=AN$+B$
1550 A$=LEFT$(A$,R-1)+RIGHT$(A$,LEN(A$)-R)
1560 NEXT X
```

Τώρα που έχουν γραφτεί δύο ενότητες χρειάζεται ένας τρόπος να τις καλούμε όταν και αν απαιτούνται. Αυτό

πετυχαίνεται, αν μεταχειριστούμε τις ενότητες σαν υπορουτίνες και να τις καλούμε, όταν απαιτείται. Το τμήμα κλήσης γίνεται με την εντολή της BASIC "GOSUB" που ουσιαστικά σημαίνει: "πήγαινε στην υπορουτίνα, αρχίζοντας από το δεδομένο αριθμό εντολής". Δοκιμάστε το κατευθείαν, πληκτρολογώντας την απευθείας εντολή:

```
GOSUB 1000
```

Τι έγινε; ο υπολογιστής θα πρέπει να είχε εκτελέσει την ενότητα 1 και να είχε διαλέξει μια λέξη από τον κατάλογο των δεδομένων (DATA). Ωστόσο, αυτό δεν συνέβη. Ο Amstrad εμφάνισε το παρακάτω μήνυμα:

```
Array already dimensioned in 1000
```

Είχε κι όλας δοθεί η διάσταση του πίνακα, όταν τρέξαμε το πρόγραμμα προηγουμένως. Η εντολή *RUN* λέει στον υπολογιστή να ξεκάσει τις τιμές όλων των μεταβλητών, συμπεριλαμβανομένων και των πινάκων, πριν ξεκινήσει την εκτέλεση του προγράμματος. Αυτή τη φορά, ωστόσο το πρόγραμμα δεν έτρεξε (RUN) αλλά εκτελέστηκε από μια

```
'GOSUB 1000'
```

Αυτή, δε δίνει αρχικές τιμές σε καμμιά μεταβλητή και έτσι ο υπολογιστής σας λέει, ότι η διάσταση του πίνακα WORD$ έχει κιόλας οριστεί.

Για να παρακάμψουμε αυτό το πρόβλημα, πληκτρολογήστε απλά:

```
'GOSUB 1100'
```

Αυτό θα εκτελέσει το πρόγραμμα εκτός από τη γραμμή 1000. Αφού πληκτρολογήσετε την παραπάνω γραμμή, θα δείτε να εμφανίζεται το μήνυμα Ready. Νομίζετε ότι ο Amstrad έτρεξε την πρώτη ενότητα και τελείωσε, αλλά στην ουσία θα έχει κάνει περισσότερα απ' αυτό. Όταν θα έχει τρέξει την ενότητα 1, θα έχει τρέξει (RUN) απευθείας στην ενότητα 2, πραγματοποιώντας την τυχαιοποίηση του AN$. Η εκτέλεση της ενότητας 2 ήταν ασχεδίαστη και έτσι επομένως εκτός ελέγχου. Για ν' ανακτήσουμε τον έλεγχο, χρειάζεται να το πούμε στον υπολογιστή όταν η υπορουτίνα τελειώσει, μέσω του συνδυασμού:

4.3.3 GOSUB...RETURN

Αυτός ο συνδυασμός κατευθύνει ένα πρόγραμμα σε μια υπορουτίνα μέσω της GOSUB και τη στέλνει πίσω. Όταν συναντήσει τη RETURN. Το πρόγραμμα επιστρέφει στην αμέσως επόμενη εντολή μετά την GOSUB.

Για να συμπεριλάβουμε τις απαιτούμενες αλλαγές, τροποποιήστε τις δύο ενότητες, προσθέτοντας τις γραμμές.

```
1040 RETURN
1170 RETURN
1600 RETURN
```

Οι δύο ενότητες μπορούν να χρησιμοποιηθούν τώρα σε μια πραγματικά δομημένη μορφή, καλούμενες από ένα πρόγραμμα ελέγχου των ενοτήτων, ξεκινώντας από τη γραμμή 500. Δηλαδή:

ΠΡΟΓΡΑΜΜΑ 4.12

```
460 GOSUB 1000:REM SET UP THE ARRAY
510 GOSUB 1100:REM PICK A WORD
520 GOSUB 1500:REM SCRAMBLE IT
640 END
```

Αφού η λέξη έχει ανακατευτεί, και ο αναγραμματισμός επιτεύχθηκε, μπορεί να εκτυπωθεί στην οθόνη. Αυτό γίνεται στη γραμμή 530 του προγράμματος 4.12(a)

ΠΡΟΓΡΑΜΜΑ 4.12(a)

```
530 LOCATE 1,6:PRINT"THE ANAGRAM IS ";AN$
```

4.4 Ενότητα 3: Εισάγοντας τη μάντεψη

Αυτή η ενότητα δεν θα έπρεπε να παρουσιάσει προφανώς μεγάλα προβλήματα, καθώς φαίνεται ότι μια απευθείας εντολή INPUT θα μπορούσε να το αντιμετωπίσει. Ωστόσο, είναι αναγκαίο κατά τη διάρκεια της εισαγωγής των δεδομένων να ελέγξουμε αν υπάρχουν εμφανή λάθη. Είναι πολύ ευκολότερο να γίνει αυτό όταν υπάρχει πρόσβαση στους χαρακτήρες, παρά όταν έχουν γίνει όλοι δεκτοί με μια απλή εντολή INPUT. Γι αυτό το λόγο, μια ρουτίνα εισόδου θα δημιουργηθεί χρησιμοποιώντας την εντολή INKEY$. Με αυτό το μέσο, κάθε χαρακτήρας μπορεί να ελεγχθεί καθώς πληκτρολογείται και το τέλος της σειράς εισόδου θα

εξεταστεί, όταν ο χρήστης πατήσει το πλήκτρο RETURN. Το πρόβλημα είναι πως θα ελεγχθεί ότι πατήσαμε το πλήκτρο ENTER. Αντίθετα από τα περισσότερα πλήκτρα του πληκτρολογίου Amstrad, το πλήκτρο ENTER δεν εμφανίζει χαρακτήρα στην οθόνη όταν πιεστεί. Για να ελέγξουμε το πάτημα του πλήκτρου ENTER χρειαζόμαστε μια καινούργια εντολή.

4.4.1 CHR$

Κάθε πλήκτρο στο πληκτρολόγιο του Amstrad έχει έναν ειδικό αριθμό χαρακτήρα που ο υπολογιστής το χρησιμοποιεί για να το αναγνωρίσει. Αυτός ο αριθμός αναφέρεται σαν η τιμή του ASCII του. ASCII είναι τ' αρχικά του American Standard Code for information Interchange. Μια από τις εντολές που χρησιμοποιούνται σε σχέση με τιμές του ASCII είναι η CHR$. Για να επιδείξουμε την εντολή CHR$ πληκτρολογήστε τα ακόλουθα:

```
PRINT CHR$(47)
```

Ο Amstrad θα εμφανίσει /. Η εντολή CHR$ έχει πει στον υπολογιστή να εμφανίσει στην οθόνη το χαρακτήρα με τιμή 47 σε ASCII. Όπως αναφέρθηκε παραπάνω, καθε πλήκτρο έχει τη δική του τιμή ASCII, ακόμη και το πλήκτρο ENTER.

```
PRINT CHR$(13)
```

Δέκα τρία είναι η τιμή του πλήκτρου ENTER. Ο Amstrad δεν μπορεί να εκτυπώσει ENTER χωρίς να του ειπωθεί να κάνει PRINT"ENTER", το οποίο δεν είναι το ίδιο πράγμα, έτσι έκανε το καλύτερο δυνατό πράγμα - έχει πράγματι παρουσιάσει ένα ENTER. Αυτό, κάνει το δρομέα να εμφανιστεί δύο γραμμές χαμηλότερα στην οθόνη.

Για να ελέγξουμε αν το ENTER έχει πατηθεί μπορεί να χρησιμοποιηθεί η γραμμή 2020 στο πρόγραμμα 4.13.

ΠΡΟΓΡΑΜΜΑ 4.13

```
2010 Z$=INKEY$:IF Z$="" THEN 2010   Input a character=
                                    Εισάγετε ένα χαρακτήρα
2020 IF Z$=CHR$(13) THEN END        check for RETURN
                                    ελέγξτε για το RETURN
2060 GOTO 2010                      look for next input
                                    κοίταξε για το επόμενο
                                    εισαγόμενο
```

Το πρόγραμμα 4.13 θα δεχτεί μια αλφαριθμητική μεταβλητή με εισαγόμενους χαρακτήρες, αλλά δεν τους αποθηκεύει. Για να γίνει αυτό, οι ξεχωριστοί χαρακτήρες

πρέπει να προστεθούν μαζί για να σχηματίσουν τη μάντεψη, έστω C$. Μην ξεχάσετε όμως, ότι κάθε φορά που μια μάντεψη εισάγεται, πρέπει να γίνει από το τίποτε, θέτοντας τη C$ ίση με τη κενή αλφαριθμητική μεταβλητή.

ΠΡΟΓΡΑΜΜΑ 4.14

```
2000 C$=""
2040 C$=C$+Z$
```

Αν όλες οι λέξεις που πρόκειται να μαντευθούν, περιέχουν μόνο γράμματα, τότε μπορεί να γίνει ένας έλεγχος για να βεβαιωθούμε γι' αυτό. Αυτά βρίσκονται μεταξύ a και z και επομένως μπορούν να ελεγχθούν λέγοντας: Αν η Z$ δε βρίσκεται μεταξύ a και z τότε πήγαινε πίσω για άλλη μια εισαγωγή. Δηλαδή:

```
2030 IF Z$<"a" OR Z$>"z" THEN GOTO 2010
```

Η γραμμή 2030 λέει στον υπολογιστή να ελέγξει την τιμή του CHR$ του εισαγόμενου χαρακτήρα με τα "a" και "z". Αν ο χαρακτήρας εισόδου είναι μικρότερος από την τιμή του χαρακτήρα "a" ή είναι μεγαλύτερος από την τιμή του χαρακτήρα "z" τότε δεν είναι γράμμα και το εισαγόμενο αγνοείται. Επειδή ο Amstrad έχει διαφορετικές τιμές CHR$ για μικρά και κεφαλαία γράμματα, πρέπει να είστε σίγουροι ότι το CAPS LOCK είναι στην κατάσταση off, πριν εκτελέσετε αυτό το πρόγραμμα.

Όλα αυτά μαζί σχηματίζουνε μια ενότητα εισόδου - ενότητα 3:

ΠΡΟΓΡΑΜΜΑ 4.15

```
2000 G$=""
2010 Z$=INKEY$:IF Z$="" THEN 2010
2020 IF Z$=CHR$(13) THEN RETURN
2030 IF Z$<"a" OR Z$>"z" THEN 2010
2040 G$=G$+Z$
2050 LOCATE 16,8:PRINT G$
2060 GOTO 2010
```

Η γραμμή 2050 εμφανίζει ότι εισάγετε γράμμα προς γράμμα. Δώσετε μεγάλη προσοχή, καθώς πληκτρολογείτε

Αυτή, λοιπόν είναι η διαδικασία εισόδου. Ωστόσο, ακόμα χρειάζεται να συνδυάσουμε μέσω της ενότητας ελέγχου

του προγράμματος, δηλαδή όπως στο πρόγραμμα 4.16.

ΠΡΟΓΡΑΜΜΑ 4.16

```
460 GOSUB 1000:REM SET UP THE ARRAY
510 GOSUB 1100:REM PICK A WORD
520 GOSUB 1500:REM SCRAMBLE IT
530 LOCATE 1,6:PRINT"THE ANAGRAM IS ";AN$
535 LOCATE 1,8:PRINT"YOUR GUESS IS"
540 GOSUB 2000:REM INPUT A GUESS
```

'Οταν η υπόθεση, έχει εισαχθεί χρειάζεται να συγκριθεί με την αρχική λέξη και το δοσμένο κατάλληλο μήνυμα. Ας το αναπτύξουμε σαν ενότητα 4.

4.5 Ενότητα 4: Ελέγχοντας τη μάντεψη

Μία απλή σύγκριση, θα εξυπηρετήσει στον έλεγχο, αν η λέξη είναι σωστή ή όχι, δηλαδή:

```
IF C$=AA$ THEN guess is correct
```

Οι έλεγχοι αυτού του τύπου, μπορούν να χρησιμοποιηθούν, όπου έγιναν οι έλεγχοι για κάθε συνθήκη και θα δίνεται αμέσως το κατάλληλο μήνυμα, όπως στο πρόγραμμα 4.17.

ΠΡΟΓΡΑΜΜΑ 4.17

```
2510 IF C$=AA$ THEN PRINT"WELL DONE GUES
S CORRECT"
2520 IF C$<>AA$ THEN PRINT"SORRY THAT'S
NOT CORRECT! TRY AGAIN"
```

Ωστόσο, ένα από τα προβλήματα μ' αυτή την απευθείας τεχνική, είναι ότι είναι περιορισμένη στο να παράγει ένα μήνυμα αμέσως μετά την IF...THEN. Ακόμη, περισσότερο από το πρόβλημα, είναι ότι χρειάζονται περισσότερες εντολές στη δοκιμή. Για παράδειγμα, όταν η υπόθεση είναι σωστή, θα ήταν επιθυμητό να ρωτήσει τον παίκτη αν χρειζεται άλλη μια προσπάθεια.

Αυτό το πρόβλημα ξεπερνιέται, έχοντας μια ρουτίνα να ελέγχει την μάντεψη και άλλες δύο ρουτίνες να το αναφέρει. Μια ρουτίνα για τη σωστή μάντεψη και μια άλλη για τη λανθασμένη. Με σκοπό να το συγχωνέψουμε, χρειάζεται μια μέθοδος για να μεταβιβάζει πληροφορίες από τη μια ρουτίνα στην άλλη. Αυτό, γίνεται χρησιμοποιώντας

σημαίες (FLAGS). Η σημαία είναι μια ειδική μεταβλητή, που χρησιμοποιείται για να δηλώσει αν έχει εκπληρωθεί μια κατάσταση. Αν μια συνθήκη εκπληρωθεί, τότε το FLAG ορίζεται σαν '1', αλλιώς ορίζεται σαν '0'. Χρησιμοποιώντας τις σημαίες μ' αυτόν τον τρόπο, είναι πολύ εύκολο ν' αναπτύξουμε μια ρουτίνα που να ελέγχει τις υποθέσεις των παικτών.

ΠΡΟΓΡΑΜΜΑ 4.18

```
2510 IF C$=AA$ THEN F1=1
```

Η μεταβλητή AA$ περιέχει τη λέξη από την οποία δημιουργήθηκε ο αναγραμματισμός. Αν η C$ (η μάντεψη του παίκτη) είναι ίση μ' αυτήν, τότε το σημαία F1 ορίζεται ίση με '1'.

Εκτός από το ότι δοκιμάζουμε για να δούμε αν ο παίκτης έχει μαντέψει σωστά τη λέξη, αυτή η ρουτίνα μπορεί να χρησιμοποιηθεί για να ελέγξει πόσες μαντέψεις έκανε ο παίκτης. Το πρόγραμμα προσφέρει έξι δυνατότητες μάντεψης. Μία άλλη σημαία χρησιμοποιείται, όταν ελέγχεται η τιμή της COUNT. Αυτή ονομάζεται F2. Αν όλες οι μαντέψεις έχουν χρησιμοποιηθεί, τότε η F2 ορίζεται σαν '1' αλλιώς σαν '0'.

ΠΡΟΓΡΑΜΜΑ 4.18(a)

```
2520 IF COUNT=6 THEN F2=1
```

Έχοντας συμπληρώσει τις δύο δοκιμές, η ρουτίνα χρειάζεται να επιστρέψει (RETURN) στο πρόγραμμα ελέγχου, στη γραμμή 2530. Η γραμμή 2500 τοποθετεί την αρχική τιμή και των δύο σημαίων στο μηδέν (0). Αυτή η αρχή, εμποδίζει τις σημαίες να περάσουν λάθος πληροφορίες, κατά τη διάρκεια μιας δεύτερης εκτέλεσης (run) του προγράμματος.

ΠΡΟΓΡΑΜΜΑ 4.19

```
2500 F1=0:F2=0
2510 IF C$=AA$ THEN F1=1
2520 IF COUNT=6 THEN F2=1
2530 RETURN
```

Το επόμενο βήμα, είναι να ενεργήσει πάνω στην τιμή των σημαίων. Αν το F1 είναι 1 τότε η υπόθεση ήταν σωστή και χρειάζεται να ειπωθεί στον παίκτη, ότι έχει κερδίσει. Η ενότητα ελέγχου, δοκιμάζει την μάντεψη αν ήταν σωστή, τότε θα καλείται η ρουτίνα 2600. Αν η μάντεψη είναι λάθος τότε καλείται η υπορουτίνα 2800.

ΠΡΟΓΡΑΜΜΑ 4.20

```
550 GOSUB 2500:REM CHECK GUESS
560 IF F1=1 THEN GOSUB 2600:GOTO 590:REM WIN
580 IF F1=0 THEN GOSUB 2800:GOTO 535
```

Η ρουτίνα στη γραμμή 2600 χρειάζεται για να πούμε στον παίκτη ότι έχει κερδίσει και πόσες προσπάθειες χρειάστηκαν. Οι γραμμές 2610, 2620, και 2630 χρησιμοποιούν μια νέα εντολή, που ονομάζεται SPACE$. Η εντολή SPACE$ χρησιμοποιείται για να τυπώνει κανείς διαστήματα! Ο αριθμός των κενών διαστημάτων που απαιτούνται σημειώνεται με τον αριθμό μέσα σε παρενθέσεις. Στο πρόγραμμα 4.20(a) τυπώνονται 40 κενά διαστήματα. Τυπώνοντας κενά διαστήματα στις κατάλληλες θέσεις, όλα τα περιττά μηνύματα μετακινούνται από την οθόνη, πριν πούμε στον παίκτη ότι έχει νικήσει.

ΠΡΟΓΡΑΜΜΑ 4.20(a)

```
2600 REM YOU HAVE WON
2610 LOCATE 1,8:PRINT SPACE$(40)
2620 LOCATE 1,12:PRINT SPACE$(40)
2630 LOCATE 1,14:PRINT SPACE$(40)
2640 LOCATE 1,8
2650 PRINT"THAT IS CORRECT!"
2660 LOCATE 1,10
2670 PRINT"THAT TOOK YOU";COUNT;"ATTEMPTS"
2680 RETURN
```

Αν η μάντεψη ήταν λάθος, τότε F1=0 και το πρόγραμμα κατευθύνεται στην υπορουτίνα της λανθασμένης μάντεψης. Όταν αυτή η υπορουτίνα, θα έχει επιστρέψει, τότε το πρόγραμμα, θα ξανακατευθύνεται στη γραμμή 530 για καινούριο εισαγόμενο.

ΠΡΟΓΡΑΜΜΑ 4.20(c)

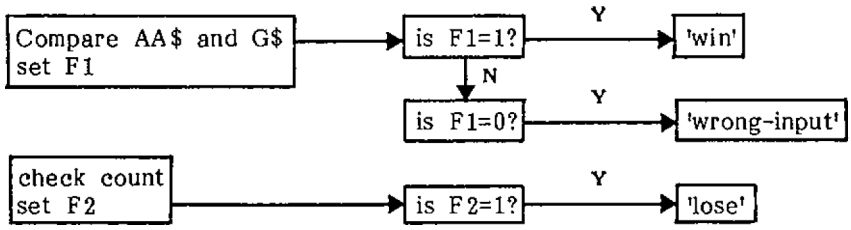
```
2800 REM INCORRECT GUESS
2810 LOCATE 1,10
2820 PRINT"I'M SORRY THAT IS WRONG"
2830 FOR X=1 TO 1000:NEXT X
2840 LOCATE 1,12
2850 PRINT"YOU HAVE HAD";COUNT;"GOES"
2860 LOCATE 1,14
2870 PRINT"YOU HAVE";6-COUNT;"TRIES LEFT"
2880 FOR X=1 TO 1000:NEXT X
2890 LOCATE 1,10:PRINT SPACE$(40)
2900 LOCATE 1,8:PRINT SPACE$(40)
2910 COUNT=COUNT+1
2920 RETURN
```

Οι γραμμές 2890 και 2900 χρησιμοποιούν το SPACE$ για να διώξουν την μάντεψη του παίκτη, και το μήνυμα 'I'M SORRY THAT IS WRONG'

Το τελευταίο μέρος της τέταρτης ενότητας ελέγχει για να δει, κατά πόσο όλες οι προσπάθειες έχουν εξαντληθεί. (Δηλαδή F2=1). Αν συμβαίνει αυτό, τότε χρειάζεται να πούμε στον παίκτη ότι έχει χάσει και ποια ήταν η λέξη, που προσπαθούσε μανιωδώς να μαντέψει.

ΠΡΟΓΡΑΜΜΑ 4.20(a)

```
570 IF F2=1 THEN GOSUB 3000:GOTO 590:REM LOST
3000 REM YOU HAVE LOST
3010 LOCATE 1,10:PRINT SPACE$(40)
3020 LOCATE 1,12:PRINT SPACE$(40)
3030 LOCATE 1,14:PRINT SPACE$(40)
3040 LOCATE 1,8
3050 PRINT"I'M SORRY YOU HAVE LOST"
3060 LOCATE 1,10
3070 PRINT"THE WORD WAS ";AA$
3090 RETURN
```



Module 4

Εικόνα 4.11

Το σημαντικότερο κομμάτι του προγράμματος, είναι η ρουτίνα ελέγχου. Που είναι έως εδώ η εξής:

ΠΡΟΓΡΑΜΜΑ 4.21

```
460 GOSUB 1000:REM SET UP THE ARRAY
510 GOSUB 1100:REM PICK A WORD
520 GOSUB 1500:REM SCRAMBLE IT
530 LOCATE 1,6:PRINT"THE ANAGRAM IS ";AN$
535 LOCATE 1,8:PRINT"YOUR GUESS IS"
540 GOSUB 2000:REM INPUT A GUESS
550 GOSUB 2500:REM CHECK GUESS
560 IF F1=1 THEN GOSUB 2600:GOTO 590:REM WIN
570 IF F2=1 THEN GOSUB 3000:GOTO 590:REM LOSE
580 IF F1=0 THEN GOSUB 2800:GOTO 535
```

Οι γραμμές 500 μέχρι την 560 καλούν όλες τις ρουτίνες και δοκιμάζουν όλες τις συνθήκες. Αυτή είναι η καρδιά του προγράμματος. Όλα όσα χρειάζεται να προστεθούν, είναι οι κανόνες και μια δοκιμή, για να δούμε κατά πόσο είναι αναγκαία άλλη μία προσπάθεια. Ακόμη η τιμή της COUNT, χρειάζεται να τεθεί στο μηδέν στην αρχή του προγράμματος.

Στον δομημένο προγραμματισμό, οι κανόνες του παιχνιδιού, θα έπρεπε να συμπεριληφθούν σε μια δικιά τους μικρή υπορουτίνα, την οποία ο υπολογιστής καλεί πριν απ' όλες τις άλλες. Αυτό κάνει το πρόγραμμα 4.22.

ΠΡΟΓΡΑΜΜΑ 4.22

```
450 GOSUB 3500:REM THE RULES

3500 REM THE RULES
3510 MODE 1
3520 LOCATE 12,2
3530 PRINT"**ANAGRAM GAME!**"
3540 LOCATE 12,4
3550 PRINT"HERE ARE THE RULES"
3560 LOCATE 8,8
3570 PRINT"I WILL THINK OF A WORD THEN"
3580 LOCATE 8,10
3590 PRINT"I WILL JUMBLE UP THE LETTERS"
3600 LOCATE 7,12
3610 PRINT"YOU MUST TRY TO GUESS THE WORD"
3620 LOCATE 5,18
3630 PRINT"**PRESS THE SPACE BAR TO BEGIN**"
3640 IF INKEY$<>" " THEN 3640
3650 RETURN
```

Έχοντας εμφανίσει τους κανόνες τώρα, να θέσουμε την COUNT στην αρχική της μηδενική τιμή, (γραμμή 470). Οι γραμμές 480 και 490 εμφανίζουν τον τίτλο του παιχνιδιού ακόμη μια φορά, έτσι ώστε [illegible] παιχνίδι που παίζεις.

ΠΡΟΓΡΑΜΜΑ 4.23

```
470 CLS:COUNT=1
480 LOCATE 12,2
490 PRINT"**ANAGRAM GAME**"
```

Το τελικό κομμάτι αυτού του παιχνιδιού, είναι να ρωτήσει τον παίκτη, αν χρειάζεται άλλη προσπάθεια. Αυτό, γίνεται μόνο αν έχουν μαντέψει σωστά, ή αν όλες οι υποθέσεις ξαντλήθηκαν. Έτσι τοποθετείται στις γραμμές 590 έως 630.

ΠΡΟΓΡΑΜΜΑ 4.24

```
590 LOCATE 4,12
600 INPUT "DO YOU WANT ANOTHER GO (Y/N)";B$
610 IF B$="Y" THEN 470
620 CLS:LOCATE 14,12
630 PRINT"GOODBYE"
640 END
```

4.5.1 LOWER$

Αυτή τη στιγμή, το παιχνίδι δουλεύει αρκετά καλά, αλλά μ' ένα μικρό εκνευρισμό, ο χρήστης πρέπει να συγουρευτεί, ότι ο υπολογιστής δουλεύει με μικρά γράμματα, πριν τρέξει το πρόγραμμα. Αυτό γιατί η γραμμή 2030 δοκιμάζει κάθε εισαγόμενο χαρακτήρα να δει αν η τιμή της σε ASCII είναι μικρότερη ή μεγαλύτερη από την τιμή σε ASCII "a" και "z" αντίστοιχα. Αν ο Amstrad είναι σε μορφή κεφαλαίων γραμμάτων, τότε θα μπορούσαν ν' αγνοηθούν οι χαρακτήρες των γραμμάτων (ο Amstrad έχει διαφορετικούς κώδικες ASCII για "A" και "a"). Η βοήθεια είναι κατά κάποιον τρόπο στη μορφή της εντολής LOWER$. Αυτή, μετατρέπει μια αλφαριθμητική μεταβλητή σε μικρά γράμματα. Για παράδειγμα:

```
A$="SMALL LETTERS"
PRINT LOWER$(A$)
```

Αυτό, θα τυπώσει την A$ με μικρά γράμματα. Αν η αλφαριθμητική περιέχει κιόλας μικρά γράμματα, τότε όλα καλά, τίποτε δεν θ' αλλάξει.

Η LOWER$ μπορεί να συγχωνευτεί στο πρόγραμμα, προσθέτοντας τη γραμμή 2025. Αυτή, μετατρέπει κάθε χαρακτήρα εισόδου σε μικρά γράμματα, έτσι τώρα δεν πειράζει αν ο υπολογιστής λειτουργεί με μικρά ή μεγάλα γράμματα.

ΠΡΟΓΡΑΜΜΑ 4.25

```
2025 Z$=LOWER$(Z$)
```

4.5.2 UPPER$

Αυτή, αν δεν το έχετε ήδη μαντέψει, μετατρέπει τις αλφαριθμητικές μεταβλητές σε κεφαλαία γράμματα. Για παράδειγμα:

```
A$="big letters"
PRINT UPPER$(A$)
```

Αυτό θα εμφανίσει την A$ με κεφαλαία.

Αυτά, καλύπτουν την εισαγωγή στο δομημένο κατά ενότητες προγραμματισμό, αφήνοντας στο χρήστη ένα απόλυτα δομημένο πρόγραμμα.

ΚΕΦΑΛΑΙΟ 5

5.1 Αρχή-τέλος προγραμματισμός: Το παιχνίδι της κρεμάλας

Υπάρχουν πολλοί διαφορετικοί τρόποι δόμησης ενός προγράμματος και καθένας έχει και τους οπαδούς του. Αυτοί που πιστεύουν σε ένα τρόπο, φροντίζουν να ενεργούν πάντοτε σύμφωνα μ' αυτόν, με σχεδόν θρησκευτικό πάθος - μια ατράνταχτη πίστη. Σ' αυτό το κεφάλαιο, θ' ακολουθηθεί μια συγκεκριμένη προσέγγιση, αλλά δε διεκδικεί την καθολική αυτοδυναμία της. Είναι, μόνο ένας από τους πολλούς τρόπους που υπάρχουν. Ωστόσο, είναι μια τεχνική, που πολλοί νιώθουν ότι είναι ιδιαίτερα πολύτιμη.

Αλλά πρώτα...

Κάθε φορά που γράφετε ένα πολύπλοκο πρόγραμμα, πρέπει από την αρχή να σημειώνονται ορισμένα πράγματα στο χαρτί. Ορισμένες ερωτήσεις πρέπει να ερωτηθούν και αυτές είναι:

* Τι πρέπει να πετύχει ένα πρόγραμμα;

* Πώς θ' αλληλεπιδράσει με το χρήστη, δηλαδή, τι εισαγόμενα θα χρειαστούν από το χρήστη και τι εξαγόμενα θα παράγουν αυτά;

* Τι στρατηγική θα χρησιμοποιηθεί για να πραγματοποιηθούν οι αναγκαίες διαδικασίες;

Στην περίπτωση, του παιχνιδιού της κρεμάλας, αυτές δεν είναι πραγματικά φοβερά δύσκολες ερωτήσεις και θα είναι πιθανότατα πιο αρμόζουσες, όταν εμπλέκονται μεγάλα πολύπλοκα συστήματα. Ωστόσο...

* Το πρόγραμμα θα ξεκινήσει με σκοπό να πετύχει έναν αλληλαπαντούμενο διάλογο, μεταξύ του υπολογιστή που παράγει μια τυχαία λέξη και του χρήστη που πρέπει να μαντέψει ποια είναι αυτή.

* Η αλληλαντίδραση μεταξύ του χρήστη και του υπολογιστή θα γίνει μέσω του πληκτρολογίου και της οθόνης.

* Το πρόγραμμα, μπορεί να διαιρεθεί, σε δύο σημαντικά τμήματα. Το πρώτο τμήμα, είναι η ρουτίνα ελέγχου (μέρος του προγράμματος που καλεί τις υπορουτίνες) και το δεύτερο τμήμα του προγράμματος, θα είναι οι ποικίλες υπορουτίνες του.

* Αναπτύξτε μια τυχαία λέξη που πρέπει να μαντέψετε.

* Συγκρίνετε τον υποτιθέμενο χαρακτήρα με τη λέξη.

Όταν ένα πρόγραμμα αναπτύσσεται, χρησιμοποιώντας τον Αρχή-Τέλος προγραμματισμό καθορίζεται πρώτα η γενική δομή και μετά το κατάλληλο πρόγραμμα, γράφεται με προοδευτικά μεγαλύτερες λεπτομέρειες. Έτσι αποφάσεις που παίρνονται νωρίς επηρεάζουν αργότερα τμήματα του προγράμματος. Αν αυτή η διαδικασία αντιστραφεί και οι λεπτομέρειες γίνουν πρώτες, τότε πολλές αλλαγές θα χρειαστούν στα λεπτομερή μέρη του προγράμματος καθώς θα καθορίζεται η δομή. Ωστόσο, πριν προχωρήσουμε σ' αυτή τη δομή, ας ρίξουμε μια ματιά σε πολύ γενικούς όρους του προγράμματος σαν σύνολα. Αυτό που θα κάνουμε, είναι να γράψουμε κάτι σαν ένα "ψευτοκώδικα". Αυτό, σημαίνει κατά ένα τρόπο κάτι που είναι λίγο σαν Αγγλικά και λίγο σαν γλώσσα υπολογιστή.

Τι θα πρέπει να κάνει το πρόγραμμα:

* Να εμφανίσει μια σελίδα με τίτλους και κανόνες. (Ενότητα 1).

* Να διαλέξει μια τυχαία λέξη διαβάζοντας από εντολές DATA ονομάζοντας τη λέξη A$. Δημιουργήστε ένα πίνακα που ονομάζεται WORD$(), που θα έχει μία τελεία για κάθε γράμμα στην A$, δηλαδή στην αρχή:

```
IF A$="COMPUTER"then WORD$()="........"
```

Όταν ένα 'M' μαντευθεί, η WORD$() θα μπορούσε να γίνει "..M....." δηλαδή το M θα μπορούσε να τοποθετηθεί στη σωστή θέση (Ενότητες 2 και 8).

* Λέει στον παίκτη, πόσα είναι τα γράμματα που πρέπει να μαντευθούν (ενότητα 3).

* Να εισαχθεί μια μάντεψη από τον παίκτη: Να ονομαστεί, ο εισαγόμενος χαρακτήρας GUESS (ενότητα 4).

* Να ελεγχθεί αν ο χαρακτήρας που μαντεύθηκε, είναι στη λέξη που θα μαντευθεί (A$). Αν είναι, τότε αντικαταστείστε την κατάλληλη τελεία στη WORD$() με αυτό το γράμμα (ενότητες 7 και 8).

* Θα αποθηκεύσει έναν κατάλογο με όλους τους χαρακτήρες που έχουν μαντευθεί ως εδώ. Ονομάστε αυτή την αλφαριθμητική μεταβλητή (string) X$ (ενότητα 5). Ελέγχει αν ο πρόσφατα μαντεμένος

χαρακτήρας έχει μαντευθεί προηγουμένως π.χ. είναι στην αλφαριθμητική με δείκτη X$? (Ενότητα 5).

* Λέει στο χρήστη αν έχει μαντευθεί προηγουμένως ο χαρακτήρας (ενότητα 6).

* Να εμφανίσει την τωρινή κατάσταση της μαντεμένης λέξης (ενότητα 9).

* Να ελέγχει για να δει αν η λέξη μαντεύθηκε. Αυτό πετυχαίνεται βλέποντας αν έμεινε καμμία τελεία στον πίνακα WORD$. Αν δεν έμεινε καμμία τελεία, τότε όλα τα γράμματα έχουν μαντευθεί (ενότητα 10).

* Αν ο παίκτης έχει νικήσει, τότε πρέπει να του δωθούν συγχαρητήρια και να ρωτηθεί αν θέλει άλλη μία προσπάθεια (ενότητα 11 και 12).

* Όταν μια λανθασμένη μάντεψη έχει γίνει, η μεταβλητή 'E' (δηλαδή της λανθασμένης μάντεψης) χρειάζεται ν' αυξηθεί κατά ένα (ενότητα 13).

* Μόλις γίνει μια λανθασμένη μάντεψη, το επόμενο τμήμα του κρεμασμένου πρέπει να ζωγραφιστεί.

* Ένας έλεγχος γίνεται στο μετρητή της λανθασμένης μάντεψης. Αν το E είναι ίσο με το 10 τότε ο παίκτης χάνει και ο άνθρωπος κρεμιέται (ενότητα 15).

5.2 Δομή προγράμματος

Πρώτα θ' αναπτυχθεί το πρόγραμμα σε μια δομή σκελετού, έτσι ώστε να τρέχει σε πολύ γενικές γραμμές, αλλά χωρίς όλες τις λεπτομέρειες. Αφού γίνει αυτό, μπορεί να ελεγχθεί η δομή και να συμπληρωθούν οι λεπτομέρειες μόνο όταν αυτή είναι σωστή.

Το πρόγραμμα αποτελείται από 16 "κολλημένες" μεταξύ τους με ένα γενικό πρόγραμμα ελέγχου ενοτήτων ώστε τώρα να μπορείτε να εργαστείτε, για κάθε ενότητα μία προς μία. Οι ποικίλες ενότητες προγράμματος θα τοποθετηθούν στο πρόγραμμα ως εξής:

Ενότητα	Γραμμές
'Ελεγχος προγράμματος	0-
Βοηθητικές λειτουργίες	900-
Ενότητα 1	1000-
Ενότητα 2	2000-
Ενότητα 3	3000-
Ενότητα 4	4000-
Ενότητα 5	5000-
Ενότητα 6	6000-
Ενότητα 7	7000-
Ενότητα 8	8000-
Ενότητα 9	9000-
Ενότητα 10	10000-
Ενότητα 11	11000-
Ενότητα 12	12000-
Ενότητα 13	13000-
Ενότητα 14	14000-
Ενότητα 15	15000-
Ενότητα 16	16000-
Εντολές δεδομένων	17000-

Εικόνα 5.1

Αυτές οι υπορουτίνες θα γραφτούν αρχικά σαν ρουτίνες δοκιμής, με σκοπό να δοκιμάσετε τη λογική ροή του όλου προγράμματος. Κάνοντάς το αυτό μπορούν να εξεταστούν όλοι οι διαφορετικοί δρόμοι του προβλήματος και να εξαφανιστούν όποια προβλήματα προκύψουν.

5.2.1 Ενότητα 1: Αρχικές συνθήκες

Χρειάζονται για να γίνει αρκετό νοικοκύρεμα. 'Οταν ξεκινά ένα πρόγραμμα, πρέπει να δοθούν γενικές πληροφορίες χρήσης, να οριστούν οι μεταβλητές κ.λ.π. Ωστόσο, για τώρα, η ενότητα 1 απλά θα καθαρίσει την οθόνη και μετά θα περιμένει να πατήσει κάποιο πλήκτρο ο χρήστης. Καθώς κάθε μια απ' τις δοκιμαστικές ενότητες θα εμφανίσουν μια οθόνη και μετά θα περιμένουν να εισαχθεί ένας χαρακτήρας, η ρουτίνα "INKEY$" θα γραφεί μια φορά στην 900 και θα κληθεί όποτε χρειαστεί από ένα 'GOSUB', δηλαδή το πρόγραμμα 5.

ΠΡΟΓΡΑΜΜΑ 5

```
900 A$=INKEY$:IF A$="" THEN 900
910 RETURN
```

ΠΡΟΓΡΑΜΜΑ 5.1

```
1000 CLS
1010 LOCATE 10,10
1020 PRINT"INITIALISATION"
1030 GOSUB 900
1040 RETURN
```

Η υπορουτίνα καλείται μέσω της ενότητας Προγράμματος Ελέγχου (PROGRAM CONTROL MODULE) (PCM) που φαίνεται στο πρόγραμμα 5(a).

ΠΡΟΓΡΑΜΜΑ 5(a)

```
500 GOSUB 1000:REM INITIALISATION
```

'Οταν εκτελεστεί θα καθαρίσει η οθόνη και μετά θα εμφανιστεί η λέξη INITIALIZATION. Μετά το πρόγραμμα, θα σταθεί και θα περιμένει μέχρι να πατηθεί ένα πλήκτρο και μετά..... Λοιπόν, δοκιμάστε το! Τρέξτε το πρόγραμμα και πατήσετε την μπάρα των διαστημάτων δύο φορές, η οθόνη θα πρέπει τώρα να δείχνει:

```
Unexpected RETURN in 910
```

Μπορείτε να δείτε τι συμβαίνει και να το εμποδίσετε να ξαναγίνει. Το πρόβλημα είναι, ότι, όπως το PCM έτρεξε και επέστρεψε από την ενότητα 1, αυτό ξανάτρεξε στην 'A$=INKEY$' ρουτίνα στη γραμμή 900. Το δεύτερο πάτημα της μπάρας των διαστημάτων, έφερε το πρόγραμμα στη γραμμή 900 και συνάντησε το 'RETURN'. Αυτή τη φορά ωστόσο, το πρόγραμμα δεν είχε που να επιστρέψει διότι δεν κλήθηκε η ρουτίνα από GOSUB, επομένως, αναφέρθηκε λάθος. Το πρόβλημα μπορεί να αποφευχθεί έξυπνα, τερματίζοντας το PCM μέσω μιας END δηλαδή, όπως στο πρόγραμμα 5(a1).

ΠΡΟΓΡΑΜΜΑ 5(a1)

```
500 GOSUB 1000: REM INITIALISATION
899 END
```

Τώρα, όταν αυτό εκτελεστεί θα εμφανιστεί μια φορά η λέξη INITIALIZATION, και αφού πιέσετε ένα πλήκτρο, το πρόγραμμα θα σταματήσει.

Ως εδώ καλά! Ωστόσο το πρόγραμμα δεν είναι πολύ φιλικό στο χρήστη, καθώς ενώ περιμένει για τα εισαγόμενα ο χρήστης δεν μπορεί να είναι βέβαιος για το τι συμβαίνει. Θα μπορούσε να ήταν πολύ σαφέστερο, να ειπωθεί στο χρήστη να πιέσει ένα πλήκτρο. Έτσι, με τη συχνώνευση ενός μηνύματος θα μπορούσε να βελτιωθεί η ρουτίνα εισόδου. Καθώς η υπορουτίνα πρόκειται να χρησιμοποιηθεί σε πολλές περιστάσεις, είναι καλύτερο αν η θέση της οθόνης είναι πάντα η ίδια. Χρησιμοποιώντας την LOCATE δε δημιουργείται πρόβλημα στο να τυπώνεται το μήνυμα στην ίδια θέση κάθε φορά.

ΠΡΟΓΡΑΜΜΑ 5(a2)

```
900 LOCATE 4,20:PRINT"PRESS ANY KEY
TO CONTINUE"
910 A$=INKEY$:IF A$="" THEN 910:ELSE RETURN
```

Τώρα, όταν το πρόγραμμα εκτελείται τυπώνεται INITIALIZATION και μετά δίνεται το μήνυμα Πάτα οποιοδήποτε πλήκτρο (PRESS ANY KEY).

Απ' εδώ, το πρόγραμμα ακολουθεί μόνο ένα δρόμο, τον:

5.2.2 Διαλέξτε λέξη

Σ' αυτό το στοιχείο, η λέξη θα επιλεχθεί απ' αυτές που είναι διαθέσιμες.

ΠΡΟΓΡΑΜΜΑ 5.2

```
2000 CLS:LOCATE 10,10
2010 PRINT"CHOOSE WORD"
2020 GOSUB 900
2030 RETURN
```

Η ενότητα καλείται από τη γραμμή 510 του PCM, όπως φαίνεται στο πρόγραμμα 5(b).

ΠΡΟΓΡΑΜΜΑ 5(b)

```
510 GOSUB 2000:REM CHOOSE WORD
```

2.2 Ενότητα 3: Εισαγωγή μάντεψης

Η ενότητα αυτή θα τοποθετήσει τη μορφή της οθόνης ου θα χρησιμοποιηθεί για το υπόλοιπο πρόγραμμα. Με λίγα όγια, μοιάζει μ' αυτό:

'ΟΓΡΑΜΜΑ 5.3

```
3000 CLS:LOCATE 10,10
3010 PRINT"DISPLAY SCREEN"
3020 GOSUB 900
3030 RETURN
```

Η ενότητα καλείται από το PCM που φαίνεται στη αμμή 520 του προγράμματος 5(c).

'ΟΓΡΑΜΜΑ 5(c)

```
520 GOSUB 3000: REM DISPLAY SCREEN
```

2.4 Ενότητα 4: Εμφάνιση στην οθόνη

Στο σημείο αυτό, έχινε μια εικασία από τον παίκτη ι ελέχθηκε με σκοπό ν' αποφευχθούν διάφορα λάθη.

'ΟΓΡΑΜΜΑ 5.4

```
4000 CLS:LOCATE 10,10
4010 PRINT"INPUT GUESS"
4020 GOSUB 900
4030 RETURN
```

Η ενότητα αυτή καλείται από τη χραμμή 530 του ογράμματος 5(d).

'ΟΓΡΑΜΜΑ 5(d)

```
530 GOSUB 4000: REM INPUT GUESS
```

5.2.5 Ενότητα 5: Δοκιμάστηκε προηγούμενα ο χαρακτήρας;

Σ' αυτό το στάδιο της ανάπτυξης δεν υπάρχει τίποτε πραγματικά για να συγκρίνετε, έτσι απαιτείται ένα απλό εισαγόμενο (Y/N) = (Ναι ή Οχι). 'Οπως στο παιχνίδι του αναγραμματισμού, θα χρησιμοποιήσουμε σημαίες για να περάσουμε πληροφορίες από τις ενότητες στο PCM. 'Ετσι, αν είναι σωστή η μάντεψη τίθεται μία σημαία στο −1 και αν είναι λάθος στο 0. Αν αυτοί οι αριθμοί μοιάζουν λίγο παράξενοι μην ανησυχείτε, όλα θ' αποκαλυφθούν αργότερα στο κεφάλαιο αυτό. Στην ενότητα 5, στη σημαία F1 τίθεται το −1 για το δεδομένο εισόδου Y και 0 για το N.

ΠΡΟΓΡΑΜΜΑ 5.5

```
5000 CLS:LOCATE 4,10
5010 PRINT"CHARACTER PREVIOUSLY TRIED (Y/N)?"
5020 GOSUB 900
5030 IF A$="Y" THEN F1=-1
5040 IF A$="N" THEN F1=0
5050 IF A$<>"Y" AND A$<>"N" THEN 5020:ELSE RETURN
```

Η ρουτίνα καλείται από τη γραμμή 540 στο PCM στο πρόγραμμα 5(f) παρακάτω:

ΠΡΟΓΡΑΜΜΑ 5(f)

```
540 GOSUB 5000:REM CHARACTER PREVIOUSSLY TRIED?
```

5.2.6 Ενότητα 6: Μήνυμα "χαρακτήρας δοκιμάστηκε προηγούμενα

Ο σκοπός αυτού του στοιχείου, είναι απλά να πει στον παίκτη, ότι ο τελευταία εισαγόμενος χαρακτήρας είχε προηγούμενα χρησιμοποιηθεί.

ΠΡΟΓΡΑΜΜΑ 5.6

```
6000 CLS:LOCATE 4,10
6010 PRINT"CHARACTER PREVIOUSLY TRIED"
6020 GOSUB 900
6030 RETURN
```

Αυτή η ρουτίνα καλείται μόνο όταν, προηγουμένως έχει μαντευθεί ο χαρακτήρας, δηλαδή όταν F1=−1, γραμμή 550 του

PCM. Καθώς έχει κληθεί η υπορουτίνα, το πρόγραμμα στρέφεται πίσω στη γραμμή 530 γι άλλο ένα δεδομένο εισόδου.

ΠΡΟΓΡΑΜΜΑ 5(g)

```
550 IF F1=-1 THEN GOSUB 6000:GOTO 530:
REM GO BACK FOR ANOTHER INPUT
```

5.2.7 Ενότητα 7: Υπάρχει η μάντεψη στη λέξη;

Μόλις έχει αποδειχτεί ότι η μάντεψη δεν είχε γίνει προηγουμένως πρέπει να γίνει ένας έλεγχος για τη σωστή μάντεψη και το πρόγραμμα θα κατευθυνθεί ανάλογα. Καθώς με την άλλη ενότητα δοκιμής, ενότητα 5, η διάρθρωση του προγράμματος θα επιτρέψει στη σημαία να οριστεί από το εισαγόμενο (Y/N)=(Ναι/Οχι).

ΠΡΟΓΡΑΜΜΑ 5.7

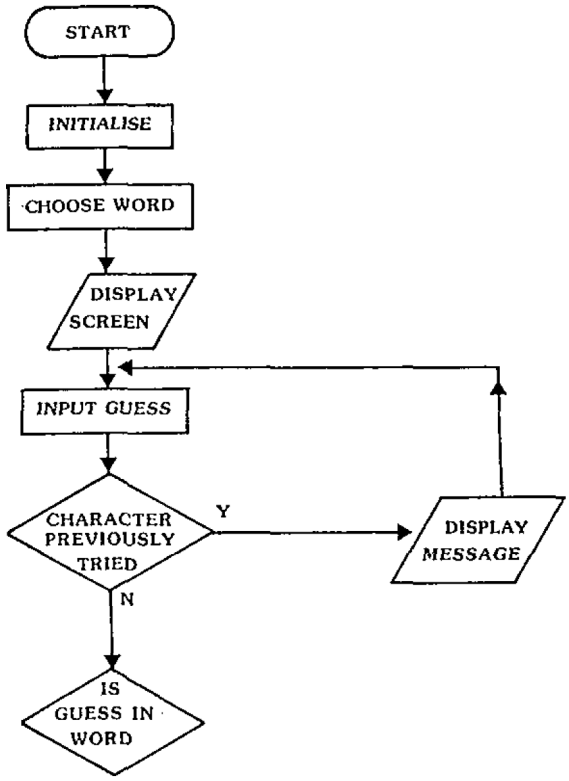
```
7000 CLS:LOCATE 4,10
7010 PRINT"IS THE GUESS IN THE WORD (Y/N)?"
7020 GOSUB 900
7030 IF A$="Y" THEN F2=-1
7040 IF A$="N" THEN F2=0
7050 IF A$<>"Y" AND A$<>"N" THEN 7020:ELSE RETURN
```

Και το PCM που το καλεί αυτό φαίνεται παρακάτω,γραμμή 560 του προγράμματος 5(h).

ΠΡΟΓΡΑΜΜΑ 5.(h)

```
560 GOSUB 7000:REM GUESS IN WORD?
```

Αφού εκτελεστεί η ενότητα αυτή θα'ναι γνωστό αν η μάντεψη είναι σωστή ή όχι και το πρόγραμμα χρειάζεται να ξανακατευθυνθεί ανάλογα. Για να γίνει αυτό σαφέστερο ας ρίξουμε μια ματιά στη ροή του προγράμματος από το λογικό διάγραμμα του σχήματος 5.2.



Εικόνα 5.2

Το γράψιμο ολόκληρου του PCM μέχρις αυτό το σημείο, δίνει το πρόγραμμα 5(i). Σ' αυτό μια δοκιμή γίνεται για το F1 που επιστρέφει το χαρακτήρα που προσπαθήσαμε προηγουμένως. 'Οταν προηγουμένως δοκιμάστηκε ο χαρακτήρας δηλαδή F1=-1 η υπορουτίνα στο 6000 καλείται να δώσει το μήνυμα δηλαδή:

```
IF F1=-1 THEN GOSUB 6000
```

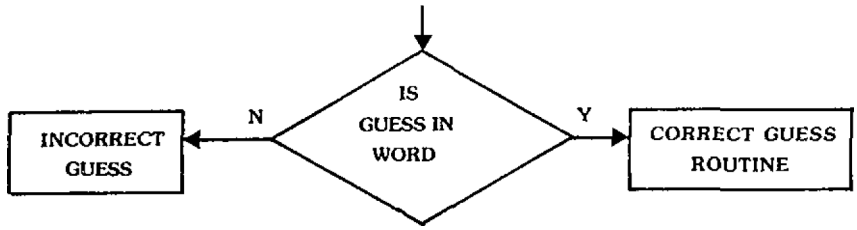
Καθώς το μήνυμα έχει δοθεί, το πρόγραμμα χρειάζεται να επιστρέψει στο σημείο του προγράμματος όπου μια άλλη μάντεψη δίνεται. Proponents δομημένου προγραμματισμού θα απαιτούσαν το πρόγραμμα, να δουλέψει μέσ' απ' όλες τις δοκιμές πριν από την επιστροφή στη γραμμή 530 με σκοπό να εισαχθεί άλλη οθόνη. Ωστόσο, proponents πραγματικού προγραμματισμού θα πρόσθεταν σε μια λογική "GOTO" σ' αυτό το σημείο! Δηλαδή:

```
550 IF F1=-1 THEN GOSUB 6000:GOTO 530
```

ΠΡΟΓΡΑΜΜΑ 5(i)

```
500 GOSUB 1000:REM INITIALISATION
510 GOSUB 2000:REM CHOOSE WORD
520 GOSUB 3000:REM DISPLAY SCREEN
530 GOSUB 4000:REM INPUT GUESS
540 GOSUB 5000:REM CHARACTER PREVIOUSLY TRIED?
550 IF F1=-1 THEN GOSUB 6000:GOTO 530
:REM GO BACK FOR ANOTHER INPUT
560 GOSUB 7000:REM GUESS IN WORD?
570 IF F2=0 THEN XXX:REM GUESS NOT IN WORD
```

'Οταν η δοκιμή στη γραμμή 550 αποτύχει το πρόγραμμα μετά δοκιμάζει ν' αποδείξει αν ήταν σωστή ή όχι η μάντεψη. 'Οπως με όλες τις δοκιμές το πρόγραμμα διακλαδίζεται ακολουθώντας την κατεύθυνση ανάλογα με το αποτέλεσμα της δοκιμής. Το σχήμα 5.3 περιγράφει αυτό στην πράξη:



Εικόνα 5.3

Για την ώρα η 'GOTO' στη γραμμή 570 θα μείνει ατέλειωτη διότι σ' αυτό το στάδιο ο αριθμός της γραμμής είναι γνωστός.

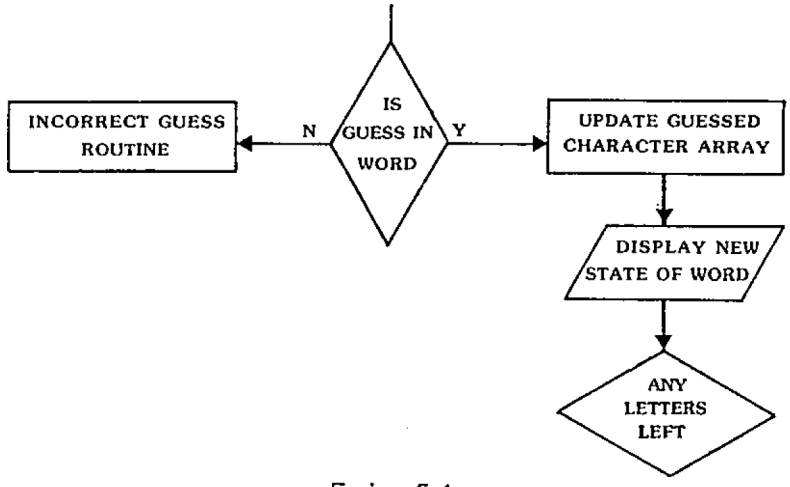
'Εως εδώ, το πρόγραμμα έχει προχωρήσει τόσο, ώστε να χειρίζεται τις συνθήκες, όπου ο χαρακτήρας που μαντεύθηκε είναι στη λέξη και ο χαρακτήρας έχει προηγουμένως δοκιμαστεί. Τώρα, η προσοχή μας στρέφεται στη περίπτωση που η υπόθεση ήταν σωστή και δεν είχε προηγουμένως δοκιμαστεί. Οι επόμενες τρεις ενότητες που εξετάζονται είναι:

8 Αποθηκεύσετε το χαρακτήρα *που πρόσφατα μαντέψατε* στον πίνακα WORD$.

9 Τροποποιήστε όσα εμφανίζονται στην οθόνη για να πείτε στο χρήστη τι συμβαίνει.

10 Ελέγξετε αν υπάρχουν άλλοι χαρακτήρες για να μαντευθούν δηλαδή αν τελείωσε το παιχνίδι.

Βάζοντας το μέσα σ' ένα τμήμα του λογικού διαγράμματος δίδει:



Εικόνα 5.4

Ιρώτα οι ενότητες...

5.2.8 Ενότητα 8: Ενημερώστε τον πίνακα των μαντεμένων χαρακτήρων

Σ' αυτήν τη ρουτίνα, ο πίνακας WORD$ χρειάζεται να ενημερωθεί με την αντικατάσταση της τελείας που αποθηκεύτηκε εκεί που έπρεπε να πάει το γράμμα με τον πραγματικό γράμμα.

ΙΡΟΓΡΑΜΜΑ 5.8

```
8000 CLS:LOCATE 10,10
8010 PRINT"UPDATE ARRAY"
8020 GOSUB 900
8030 RETURN
```

5.2.9 Ενότητα 9: Εμφάνιση νέας οθόνης

Αυτή η ενότητα που εμφανίζεται σε σχήμα σκελετού στο πρόγραμμα 5.9, απλά αναφέρει την τωρινή κατάσταση της λέξης που μαντεύτηκε.

ΙΡΟΓΡΑΜΜΑ 5.9

```
9000 CLS:LOCATE 4,10
9010 PRINT"THE WORD SO FAR IS"
9020 GOSUB 900
9030 RETURN
```

.2.10 Ενότητα 10: Έλεγχος για όλους τους μαντεμένους χαρακτήρες

Σ' αυτό το σημείο, πραγματοποιείται ένας έλεγχος
να να καθορίσει κατά πόσο, ο παίκτης μάντεψε όλους τους
αρακτήρες στη λέξη.

ΙΡΟΓΡΑΜΜΑ 5.10

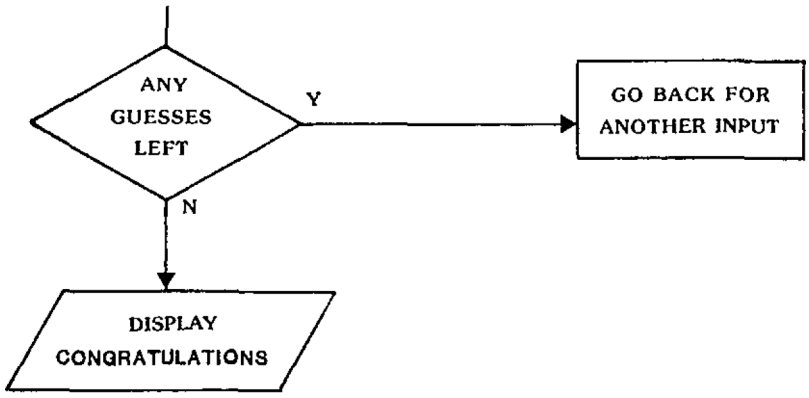
```
10000 CLS:LOCATE 4,10
10010 PRINT"ANY CHARACTERS LEFT TO GUESS (Y/N)?"
10020 GOSUB 900
10030 IF A$="Y" THEN F3=-1
10040 IF A$="N" THEN F3=0
10050 IF A$<>"Y" AND A$<>"N" THEN 10020:ELSE RETURN
```

Το PCM ως αυτό το σημείο είναι σχεδόν στρωτό, καθώς
ίει κατευθείαν διαμέσου της ενότητας 8 στη 10η ενότητα,
πως στο προγραμμα 5(j).

ΙΡΟΓΡΑΜΜΑ 5(j)

```
580 GOSUB 8000:REM UPDATE ARRAY
590 GOSUB 9000:REM DISPLAY SCREEN
600 GOSUB 10000:REM ANY CHARACTERS LEFT TO GUESS?
```

Καθώς έγινε η δοκιμή για να δει αν όλοι οι
αρακτήρες μαντεύθηκαν, τότε το πρόγραμμα αποκλίνει ακόμη
ια φορά. Αν μένουν ακόμα χαρακτήρες για να μαντευθούν
ηλαδή F3=-1 τότε το πρόγραμμα επιστρέφει για άλλο ένα
σαγόμενο (ενότητα 4) και όταν όλοι οι χαρακτήρες έχουν
αντευθεί, το παιχνίδι τελειώνει και πρέπει να ειπωθεί
το παίκτη, το σχήμα 5.5.



Εικόνα 5.5

Ο ευκολότερος από τους δρόμους αυτούς για να καταπιαστείτε είναι αυτός που πηγαίνει πίσω για ένα καινούριο εισαγόμενο. Όλα όσα απαιτούνται είναι μια επανάκαμψη του προγράμματος στη ρουτίνα εισαγομένων όταν η F3 τίθεται στο -1 δηλαδή:

```
IF F3=-1 THEN 530
```

Όταν η σημαία δεν καθορίζεται, τότε το πρόγραμμα θα καλέσει την ενότητα 11.

```
610 IF F3=-1 THEN 530
620 GOSUB 11000:REM WELL DONE
```

Μια και ο παίκτης πληροφορείτε ότι κέρδισε (ενότητα 11) η ενότητα 12 θα ρωτήσει τον παίκτη αν χρειάζεται καινούργια προσπάθεια. Έτσι οι ενότητες 11 και 12 είναι:

5.2.11 Ενότητα 11: Λέει στον παίκτη "well done"

ΠΡΟΓΡΑΜΜΑ 5.11

```
11000 CLS:LOCATE 4,10
11010 PRINT"WELL DONE YOU HAVE GUESSED
THE WORD"
11020 GOSUB 900
11030 RETURN
```

5.2.12 Ενότητα 12: Ρωτά "Do you want another go?"

ΠΡΟΓΡΑΜΜΑ 5.12

```
12000 CLS:LOCATE 4,10
12010 PRINT"DO YOU WANT ANOTHER GO (Y/N)?"
12020 GOSUB 900
12030 IF A$="Y" THEN F5=-1
12040 IF A$="N" THEN F5=0
12050 IF A$<>"Y" AND A$<>"N" THEN 12020:ELSE RETURN
```

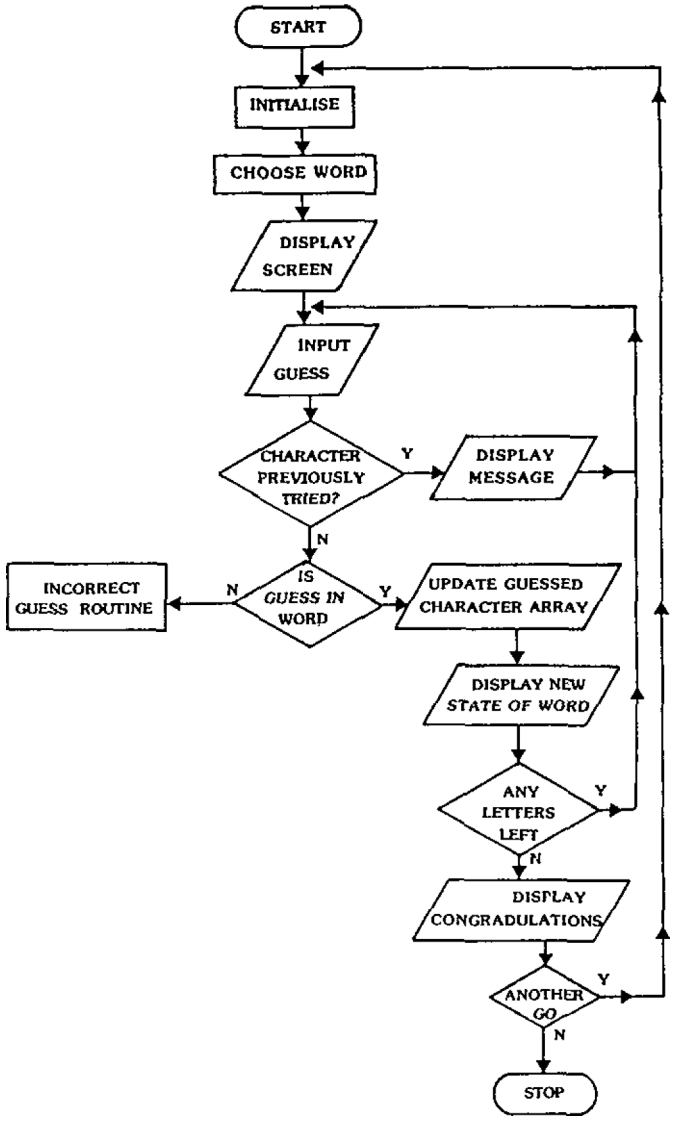
Η ενότητα 12 καλείται με μια απλή "GOSUB" αλλά τότε η επιστρεφόμενη σημαία πρέπει ν' αποκωδικοποιηθεί. Όταν χρειάζεται "άλλη μία προσπάθεια" το πρόγραμμα ξανακατευθύνεται πίσω στην ενότητα 1 και όταν δε ζητούνται άλλες προσπάθειες, ολόκληρο το πρόγραμμα τελειώνει. Το PCM που το καλεί είναι το:

ΠΡΟΓΡΑΜΜΑ 5(1)

```
630 GOTO 700:REM ANOTHER GO?
.
.
.
700 GOSUB 12000:REM ANOTHER GO?
710 IF F5=-THEN 500:REM PLAY IT AGAIN SAM!
720 CLS:LOCATE 10,10
730 PRINT "GOODBYE!"
740 END
```

Αν χρειαστεί κι άλλη προσπάθεια το πρόγραμμα κατευθύνεται στη γραμμή 500 που καλεί τη ρουτίνα INITIALIZATION έτσι ώστε το πρόγραμμα να ξαναξεκινήσει.

Το σχήμα 5.6 δείχνει το ως τώρα λογικό διάγραμμα:



Εικόνα 5.6

Από το σχήμα 5.6 μπορεί να φανεί ότι όλα αυτά που παραμένουν για να γίνουν είναι οι ρουτίνες που χειρίζονται την πορεία από την ενότητα 6 όπου F2=0 δηλαδή η μάντεψη δεν ήταν στη λέξη. Η γραμμή 570 μπορεί τώρα να συμπληρωθεί για να διαβάζει:

```
570 IF F2=0 THEN 640:REM GUESS NOT IN WORD
```

Κατά μήκος αυτής της πορείας, τα πρώτα στάδια είναι: Ν' αυξήσουμε το μετρητή της λανθασμένης μάντεψης (ενότητα 13), να εμφανίσει την καινούργια οθόνη (ενότητα 14) και να ελέγξει αν όλες οι προσπάθειες χρησιμοποιήθηκαν (ενότητα 15) Σχήμα 5.7

Εικόνα 5.7

Τώρα οι ενότητες...

5.2.13 Ενότητα 13: Αύξηση του μετρητή λανθασμένης μάντεψης

ΠΡΟΓΡΑΜΜΑ 5.13

```
13000 CLS:LOCATE 4,10
13010 PRINT"INCREMENT WRONG GUESS COUNT"
13020 GOSUB 900
13030 RETURN
```

5.2.14 Ενότητα 14: Εμφάνιση καινούριας οθόνης

ΠΡΟΓΡΑΜΜΑ 5.14

```
14000 CLS:LOCATE 4,10
14010 PRINT"DISPLAY NEW SCREEN"
14020 GOSUB 900
14030 RETURN
```

5.2.15 Ενότητα 15: Χρησιμοποιήθηκαν όλες οι προσπάθειες;

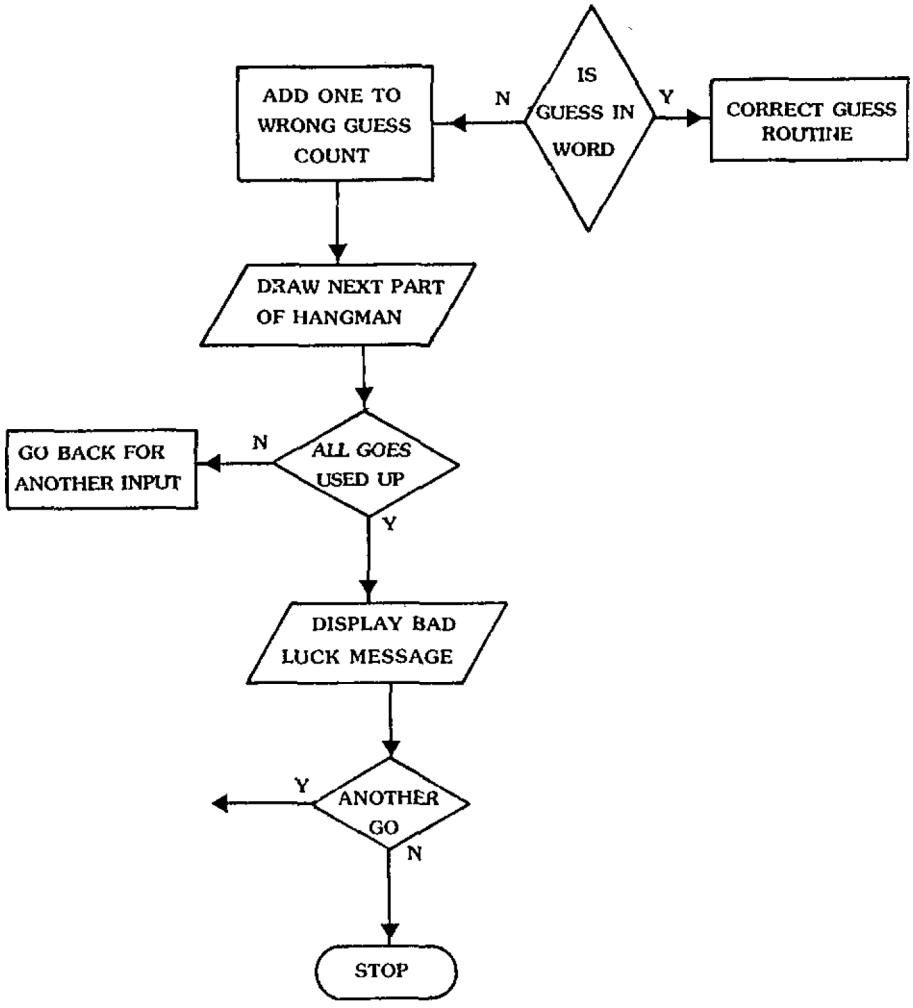
ΠΡΟΓΡΑΜΜΑ 5.15

```
15000 CLS:LOCATE 4,10
15010 PRINT"ARE ALL GOES USED (Y/N)?"
15020 GOSUB 900
15030 IF A$="Y" THEN F4=-1
15050 IF A$="N" THEN F4=0
15060 IF A$<>"Y" AND A$<>"N" THEN 15020:ELSE RETURN
```

ΠΡΟΓΡΑΜΜΑ 5(m)

```
640 GOSUB 13000:REM INCREMENT WRONG GUESS COUNT
650 GOSUB 14000:REM DISPLAY NEW SCREEN
660 GOSUB 15000:REM ARE ALL GOES USED?
```

Μόλις η ενότητα 15 εκτελεστεί πρέπει να γίνει ένας έλεγχος στην κατάσταση του F4. Αν δεν έχει οριστεί (F4=0) τότε όλες οι προσπάθειες δεν έχουν ακόμη χρησιμοποιηθεί και το πρόγραμμα ανακυκλώνεται πίσω στη γραμμή 530 για άλλο ένα εισαγόμενο. Ωστόσο, αν έχουν χρησιμοποιηθεί όλες, τότε ο παίκτης πληροφορείται και ερωτάται αν επιθυμεί άλλη προσπάθεια. Το σχέδιο 5.8 εμφανίζει όλη τη γενική κατάσταση.



Εικόνα 5.8

Αν όλες οι προσπάθειες εξαντλήθηκαν, τότε ο παίκτης κρεμιέται και ερωτάται αν απαιτείται άλλη προσπάθεια. Το πρόγραμμα αυτό απλά τρέχει στη γραμμή 700, όπου ο παίκτης ερωτάται αν θέλει άλλη προσπάθεια.

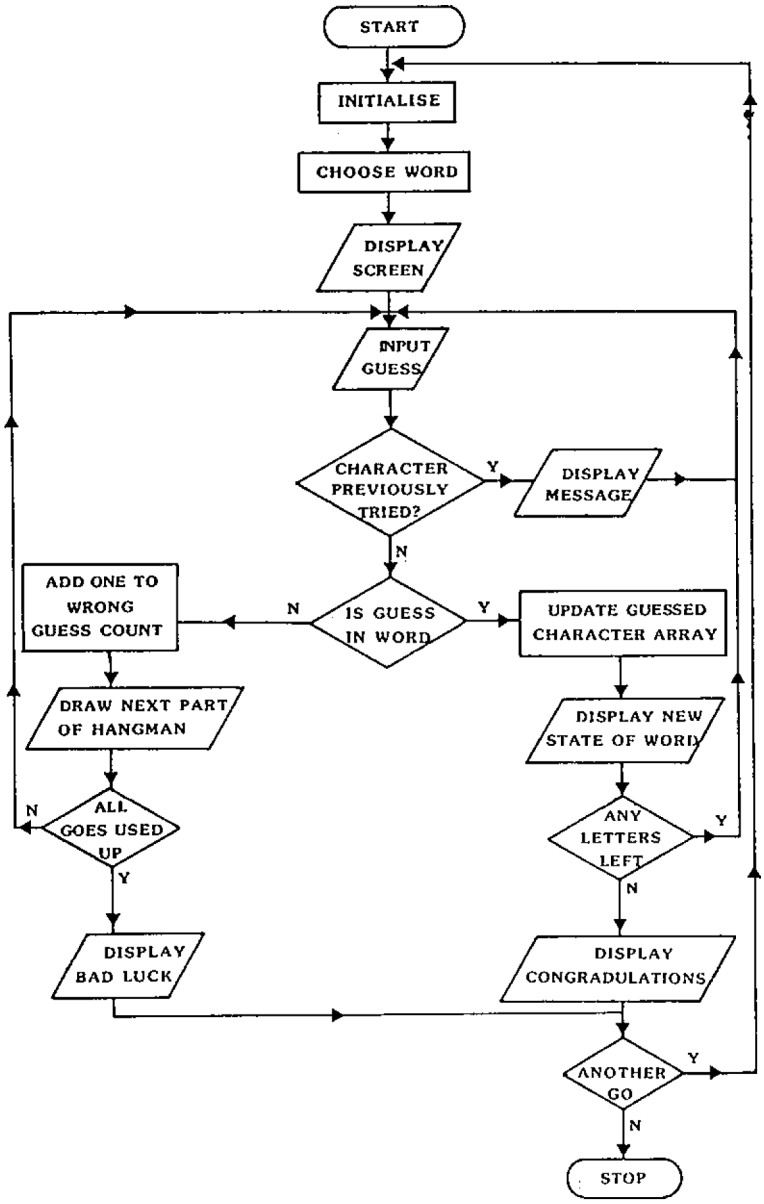
5.2.16 Ενότητα 16: Χάσατε

Μόνο η ενότητα 16 απομένει να καθαριστεί, που είναι η εκτύπωση που λέει ότι όλες οι προσπάθειες εξαντλήθηκαν δηλαδή:

ΠΡΟΓΡΑΜΜΑ 5.16

```
16000 CLS:LOCATE 4,10
16010 PRINT "ALL GOES USED UP"
16020 GOSUB 900
16030 RETURN
```

Σ' αυτό το στάδιο της πορείας ολόκληρος ο σκελετός του προγράμματος έχει γραφτεί και το λογικό διάγραμμα μπορεί να σχεδιαστεί. Το σχήμα 5.9 το δείχνει αυτό σ' όλο του το μεγαλείο.


START
INITIALISE
CHOOSE WORD
DISPLAY SCREEN
INPUT GUESS
CHARACTER PREVIOUSLY TRIED?
Y
DISPLAY MESSAGE
N
ADD ONE TO WRONG GUESS COUNT
N
IS GUESS IN WORD
Y
UPDATE GUESSED CHARACTER ARRAY
DRAW NEXT PART OF HANGMAN
DISPLAY NEW STATE OF WORD
N
ALL GOES USED UP
Y
ANY LETTERS LEFT
Y
N
DISPLAY BAD LUCK
DISPLAY CONGRADULATIONS
ANOTHER GO
Y
N
STOP

Εικόνα 5.9

Η επόμενη εργασία είναι να εξετάσουμε ότι όλες οι συνθήκες που μπορούν να συμβούν κατά τη διάρκεια μιας εκτέλεσης του παιχνιδιού έχουν προνοηθεί. Ένας τρόπος για να το αντιμετωπίσουμε αυτό είναι να δημιουργήσουμε ένα κατάλογο όλων των σημείων όπου συμβαίνουν οι διακλαδώσεις και μετά να εκτελέσουμε το πρόγραμμα επιλέγοντας μια διακλάδωση σε κάθε προσπάθεια.

Όλες οι σημαίες στο πρόγραμμα έχουν δύο συνθήκες, την "YES" και την "NO". Το σχήμα 5.10 καταγράφει όλες τις σημαίες και τις πιθανές τους καταστάσεις.

Σημαίες	Καταστάσεις	
F1	YES	NO
F2	YES	NO
F3	YES	NO
F4	YES	NO
F5	YES	NO

Εικόνα 5.10

Τώρα ας τρέξουμε το πρόγραμμα και ας περάσουμε μέσα από όλα τα δυνατά στάδια. Παρακάτω δίνεται το μήνυμα της οθόνης και με πιο σκούρα γράμματα η είσοδος που θα πρέπει να δώσετε. Θυμηθείτε ότι το "Y" πρέπει νάναι με κεφαλαία.

Στάδιο

1	Initialisation	SPACE
2	Choose word	SPACE
3	Display screen	SPACE
4	Input guess	SPACE
5	Character previously tried?	Y
6	Display 'previously tried' message	SPACE
7	Input guess	SPACE

Εικόνα 5.11

Το σχήμα 5.11 δείχνει τα στάδια που συμβαίνουν όταν το πρόγραμμα τρέξει για πρώτη φορά. Το πρώτο είναι αυτό που ελέγχει ένα χαρακτήρα που μαντεύθηκε προηγούμενα. Όταν εμφανίζεται το μήνυμα "προηγούμενα δοκιμασμένος χαρακτήρας;" Πίεσε το "Y". Τότε ο υπολογιστής θα εμφανίσει το ακόλουθο:

CHARACTER PREVIOUSLY TRIED

Προφανώς αυτό το τμήμα του ελέγχου του PCM δουλεύει. Συνεχίζοντας...

Σημαία	Καταστάσεις	
F1	YES	NO
F2	YES	NO
F3	YES	NO
F4	YES	NO
F5	YES	NO

Εικόνα 5.12

Τώρα συνεχίστε ελέγχοντας μερικές ακόμα κατευθύνσεις: Πατείστε το SPACE για να καθαρίσετε το στάδιο 7

Στάδιο

8	Was character previously tried?	N
9	is guess in word?	Y
10	Update array	SPACE
11	Display new screen (the word so far)	SPACE
12	Any characters left to guess?	Y
13	Input guess	SPACE

Εικόνα 5.13

Μία άλλη συμπληρωμένη ανακύκλωση. Ας σημειώσουμε τι έχει δοκιμαστεί ως εδώ:

Σημαία	Καταστάσεις	
F1	YES	NO
F2	YES	NO
F3	YES	NO
F4	YES	NO
F5	YES	NO

Εικόνα 5.14

Αυτές είναι μόλις οι μισές πορείες που δοκιμάστηκαν ως εδώ. Κάνετε μια προσπάθεια μόνοι στο σχεδιασμό των υπολοίπων δοκιμών. Στην περίπτωση που έχετε προβλήματα, ένας πιθανός τρόπος για να τον κάνετε δίνεται παρακάτω:

Στάδιο

14	Was character previously tried?	**N**
15	Is guess in word?	**Y**
16	Update array	**SPACE**
17	Display new screen (the word so far)	**SPACE**
18	Are characters left to guess?	**N**
19	Tell player well done	**SPACE**
20	Want another go?	**Y**
21	Initialisation	**SPACE**
22	Choose word	**SPACE**
23	Display screen	**SPACE**
24	Input guess.	**SPACE**

Εικόνα 5.15

Άλλη μια ανακύκλωση και μερικές ακόμα δοκιμές σημειώνονται.

Σημαία	Καταστάσεις	
F1	YES	NO
F2	YES	NO
F3	YES	NO
F4	YES	NO
F5	YES	NO

Εικόνα 5.16

Αυτό μόλις που αφήνει να γίνει η δοκιμή, κατά μήκος της πορείας των "λανθασμένων μαντειών". Έτσι ξαναπάμε με ένα SPACE για να καθορίσουμε το εισαγόμενο.

Στάδιο

25	Was character previously tried?	**N**
26	Is guess in word?	**N**
27	Increment wrong guess count	**SPACE**
28	Display new screen	**SPACE**
29	All goes used?	**N**
30	Input guess	**SPACE**
31	Character previously tried?	N
32	Is guess in word?	**N**
33	Increment wrong guess count	**SPACE**
34	Display new screen	**SPACE**
35	All goes used?	**Y**
36	Display screen	**SPACE**
37	Want another go?	**N**

Εικόνα 5.17

Το πρόγραμμα θα πρέπει τώρα να'χει τελειώσει. Αν το πρόγραμμα τελείωσε σ' αυτό το σημείο τότε όλα καλά, όλοι οι δρόμοι δοκιμάστηκαν με επιτυχία. Αν φαίνεται πολύ ενοχλητικό είναι μόνο γιατί έτσι είναι. Ακόμα κι ένα προφανώς εύκολο πρόγραμμα σαν κι αυτό έχει μια δικαιολογημένα περίπλοκη δομή και χρειάζεται να δοκιμαστεί ολόκληρο πριν προστεθούν οι υπορουτίνες. Αυτό είναι το επόμενο στάδιο:

5.3 Αναπτύσσοντας τις ενότητες

Αφού αναπτύχθηκε η όλη δομή του προγράμματος, δουλεύοντας μπορούν να σχεδιαστούν οι ιδιαίτερες ενότητες, να τοποθετηθούν σε θέση, μία κάθε φορά. Με τον τρόπο αυτό ολόκληρο το πρόβλημα μπορεί να κομματιαστεί σε εύχρηστα μέρη. Οι διάφορες υπορουτίνες αναπτύχθηκαν για να ευκολύνουν τη δοκιμή μέσα στη δομή του προγράμματος.

Κατ' αρχάς...

5.3.1 Ενότητα προγράμματος ελέγχου

```
500 GOSUB 1000:REM INITIALISATION
510 GOSUB 2000:REM CHOOSE WORD
520 GOSUB 3000:REM DISPLAY SCREEN
530 GOSUB 4000:REM INPUT GUESS
540 GOSUB 5000:REM CHARACTER PREVIOUSLY
TRIED?
550 IF F1=-1 THEN GOSUB 6000:GOTO 530:
REM REPORT CHARACTER PREVIOUSLY TRIED
560 GOSUB 7000:REM GUESS IN WORD?
570 IF F2=0 THEN 640:REM GUESS NOT IN W
ORD
580 GOSUB 8000:REM UPDATE ARRAY
590 GOSUB 9000:REM DISPLAY SCREEN
600 GOSUB 10000:REM ANY CHARACTERS LEFT
TO GUESS?
610 IF F3=-1 THEN 530:REM MORE CHARACTE
RS TO GUESS
620 GOSUB 11000:REM WELL DONE
630 GOTO 700:REM ANOTHER GO?
640 GOSUB 13000:REM INCREMENT WRONG GUE
SS COUNT
650 GOSUB 14000:REM DISPLAY NEW SCREEN
660 GOSUB 15000:REM REM ARE ALL GOES US
ED?
670 IF F4=0 THEN 530:REM MORE GUESSES
680 GOSUB 16000:REM YOU HAVE LOST
```

```
700 GOSUB 12000:REM ANOTHER GO?
710 IF F5=-1 THEN 500:REM PLAY IT AGAIN
 SAM!
720 CLS:LOCATE 10,10
730 PRINT"GOODBYE"
740 END
```

Θα σας άρεσε στο στάδιο αυτό να απαλλαγείτε από το περιττό END της γραμμής 899 καθώς τώρα η γραμμή 740 είναι το "επίσημο" τέλος του προγράμματος.

5.3.2 Αρχικές συνθήκες

Η ρουτίνα των αρχικών συνθηκών καθαρίζει πρώτα την οθόνη και τοποθετεί τα χρώματα, που το παιχνίδι θα χρησιμοποιήσει στα "μελανοδοχεία" από 0 έως 3. Μετά από αυτό οι γραμμές 1040–1190 χρησιμοποιούνται για να δώσουν τους κανόνες του παιχνιδιού, αυτές έχουν μείνει για σας, τον αναγνώστη, για να κάνετε ότι καλύτερο μπορείτε. Πραγματική φροντίδα πρέπει να δοθεί σ' αυτή την τεκμηριωμένη φάση, έτσι ώστε αν το πρόγραμμα είναι να σταθεί μόνο του χωρίς να πρέπει να το εξηγήσετε οι κανόνες θα πρέπει να είναι απόλυτα καθαροί.

Η ρουτίνα αρχικών συνθηκών μπορεί τώρα να γραφτεί, το τι θα κάνει είναι:

- Να καθαρίσει την οθόνη και να βάλει χρώματα (1010–1030).
- Αναγγέλει το παιχνίδι (1040).
- Δίνει τους κανόνες (1050–1190).
- Να σταματήσει το πρόγραμμα μέχρι να διαβαστούν οι κανόνες και πατηθεί ένα κουμπί (1200–1205).

Βάζοντάς τα αυτά σ' ένα πρόγραμμα:

ΠΡΟΓΡΑΜΜΑ 5.1(a)

```
1000 REM initialisation
1010 MODE 1
1020 INK 0,13:INK 1,26:INK 3,17:INK 4,3
1030 BORDER 17:PEN 1:PAPER 0:CLS
1040 LOCATE 14,2:PRINT"***HANGMAN***"
1050 LOCATE 4,6
1060 PRINT"Here are the rules!"
1070 LOCATE 4,8
1080 PRINT"There are no rules!"
1190 LOCATE 4,20
1200 PRINT"Press space bar to continue"
1205 IF INKEY$<>" " THEN 1190
```

Κάθε φορά που παίζεται το παιχνίδι, διάφορες μεταβλητές χρειάζεται να ξαναοριστούν και να καθοριστούν αλφαριθμητικές π.χ. η Z$. Στην πρώτη εκτέλεση του παιχνιδιού αυτή παίρνει την τιμή "πρώτη" δηλαδή η οθόνη λέει τότε ποια είναι η Z$ (*πρώτη* μάντεψη). Αμέσως μετά τη χρήση η Z$ επανατοποθετείται με το "NEXT", ώστε ο παίκτης να ρωτηθεί μετά "ποια είναι η Z$ (επόμενη) μάντεψη. Βέβαια μόλις το παιχνίδι έχει παιχθεί μια φορά η Z$="NEXT" χρειάζεται να ξαναοριστεί και να ξαναρχικοποιηθεί. Επίσης χρειάζεται να ξαναοριστούν και άλλες μεταβλητές, όπως η E (ο αριθμός των ως τώρα λανθασμένων μαντέψεων) και η X$ (μια αλφαριθμητική μεταβλητή που περιλαμβάνει όλες τις μαντέψεις που έχιναν ως τώρα).

ΠΡΟΓΡΑΜΜΑ 5.1(b)

```
1220 Z$=" FIRST "
1230 E=0
1240 X$=""
1250 RETURN
```

Μια και αυτό έχει εισαχθεί το πρόγραμμα μπορεί να εκτελεστεί. Ωστόσο εκτός από τη σελίδα "τίτλου" θα φανεί λίγο διαφορετικό από την προηγούμενη εκτέλεση του. Στο στάδιο αυτό δεν έχιναν βελτιώσεις στην εκτύπωση οθόνης.

5.3.3 Ενότητα 2: Επιλογή λέξης

Η ρουτίνα που δείχνεται στο πρόγραμμα 5.2(a), παράγει έναν τυχαίο αριθμό και μετά ψάχνει τόσες φορές όσες ο αριθμός ανάμεσα στις εντολές της DATA.

ΠΡΟΓΡΑΜΜΑ 5.2(a)

```
2000 REM CHOOSE WORD
2010 R=INT(RND*10)+1
2011 R=4:REM development only
2020 RESTORE
2030 FOR X=1 TO R
2040 READ A$
2050 NEXT X
2060 L=LEN(A$)
```

Η γραμμή 2011 δίνει στην "R" την τιμή "4" έτσι ώστε όταν δοκιμάσουμε το πρόγραμμα ξέρουμε τι λέξη να μαντέψουμε. Αυτό θα μας γλυτώσει πολύ χρόνο στο μοντάρισμα του προγράμματος.

Προσέξετε ότι η FOR...NEXT (ανακύκλωση) στη 2030 τρέχει από 1 μέχρι R. Δηλαδή αλλάζει τυχαία καθώς δημιουργούνται διαφορετικοί τυχαίοι αριθμοί στη γραμμή 2010.

Εκτός από την επιλογή της λέξης, η υπορουτίνα υπολογίζει επίσης το μήκος της L, καθώς αυτό χρειάζεται σ' άλλες υπορουτίνες. Η μεταβλητή "L" χρησιμοποιείται μετά για να συμπληρώσει το string WORD$. Στο ξεκίνημα του παιχνιδιού, αυτό περιέχει απλά τον αναγκαίο αριθμό των στιγμάτων - δηλαδή ένα για κάθε γράμμα. Καθώς γίνονται σωστές υποθέσεις τα κατάλληλα γράμματα εισάγονται στη σωστή θέση στην αλφαριθμητική με δείκτη και μετά εμφανίζονται στην οθόνη. Έτσι αν η επιλεγμένη λέξη (A$) είναι αρχικά η "COMPUTER", τότε L=8 και κάθε τιμή στην αλφαριθμητική με δείκτη WORD$ ορίζεται σε μια τελεία.

ΠΡΟΓΡΑΜΜΑ 5.2(b)

```
2500 WORD$=""
2510 FOR X=1 TO L
2520 WORD$(X)=".":WORD$=WORD$+"."
2530 NEXT X
2540 RETURN
```

Τα δεδομένα (DATA) που πρέπει να γίνουν READ διαβαστούν είναι είναι αποθηκευμένα στις γραμμές 18000 και πέρα όπως στο πρόγραμμα 5.2(c).

ΠΡΟΓΡΑΜΜΑ 5.2(c)

```
18000 DATA CAT,TURBINE,PLATE,COMPUTER
18010 DATA
18020 DATA
18030 DATA
```

Τα υπόλοιπα δεδομένα αφήνονται να τα γεμίσετε εσείς. Όταν η ενότητα 2 πληκτρολογηθεί και το πρόγραμμα εκτελεστεί δεν θα μπορεί να ανακαλυφθεί καμμία διαφορά, καθώς στο στάδιο αυτό δεν εκτυπώνεται τίποτα από τη δουλειά αυτή στην οθόνη.

5.3.4 Ενότητα 3: Εκτυπώσεις στην οθόνη

Στο σημείο αυτό του παιχνιδιού, όλα όσα πρέπει να εμφανιστούν είναι μόνο το μήνυμα που λέει στον παίκτη πόσα γράμματα έχει η λέξη και η αλφαριθμητική W$.

ΠΡΟΓΡΑΜΜΑ 5.3(a)

```
3000 REM display the screen
3005 CLS
3010 LOCATE 14,1:PRINT"***HANGMAN***"
3020 LOCATE 4,3
3030 PRINT"The word is"
3040 LOCATE 16,3:PEN 2
3050 PRINT word$
3060 LOCATE 4,5:PEN 1
3070 PRINT"The word has";
3080 PEN 2:PRINT l;:PEN 1
3090 PRINT"letters"
3100 RETURN
```

Τώρα όταν το ως εδώ πρόγραμμα γίνει RUN, οι αρχές του παιχνιδιού θ' αρχίσουν να αναδύονται. Ωστόσο η εκτύπωση θα εξαφανιστεί αμέσως γιατί η γρήγορη απάντηση "Πάτα ένα κουμπί" έχει μετακινηθεί και η ρουτίνα επιστρέφει αμέσως στη γραμμή 520. Και πάλι η ακριβής φύση της τελειωμένης εκτύπωσης της οθόνης εξαρτάται από εσάς.

5.3.5 Ενότητα 4: Εισαγόμενο μιας μάντεψης

Όλα όσα απαιτούνται εδώ είναι ένα απλό μήνυμα, να πει στο χρήστη να εισάγει μία υπόθεση. Μερικές φορές, ωστόσο, αυτή θα'ναι η "πρώτη" (FIRST) και μερικές η "επόμενη" (NEXT). Αυτό μπορεί να προσαρμοστεί ορίζοντας τη λέξη "πρώτη" σε μια αλφαριθμητική στο ξεκίνημα του προγράμματος και μετα, καθώς το πρόγραμμα θα'χει τρέξει, αλλάζοντας τα περιεχόμενα της μεταβλητής στο "επόμενα" (NEXT). Τα δύο στοιχεία που ενεργούν κατ' αυτόν τον τρόπο δίνονται στο πρόγραμμα 5.4(a) Η χρήση μιας INKEY$, παρά μιας INPUT, επιτρέπει την είσοδο ενός μόνο χαρακτήρα – ενδεχομένως ένα γράμμα – χωρίς να χρειάζεται να πατηθεί το ENTER.

ΠΡΟΓΡΑΜΜΑ 5.4(a)

```
1210 Z$=" FIRST "

4000 REM input a guess
4010 LOCATE 4,7
4020 PRINT"What is your";z$;"guess?     "
4025 LOCATE 29,7:PEN 3:PRINT CHR$(143):PEN 1
4030 g$=INKEY$:IF g$="" THEN 4030
4040 g$=UPPER$(g$)
4050 IF g$<"A" OR g$>"Z" THEN 4030
4060 LOCATE 29,7:PEN 2:PRINT g$
4070 z$=" next ":PEN 1
4080 RETURN
```

Ο εισαγόμενος χαρακτήρας μετατρέπεται σε κεφαλαία χρησιμοποιώντας την εντολή UPPER$.

Τώρα, όταν εκτελείται, το πρόγραμμα θα πάρει, όσο εισάγεται, μια εικασία που εκχωρείται στη μεταβλητή G$ και τυπώνεται πάνω στην οθόνη.

5.3.6 Ενότητα 5: Έχει δοκιμαστεί προηγουμένως χαρακτήρας;

Όταν ο παίκτης εισάγει μια μαντεία που είναι επανάληψη προηγούμενης εισόδου, το πρόγραμμα του φέρεται ευγενικά. Θάταν δυνατό να επιφορτίσει αυτή τη μαντεία πάνω στον αριθμό των επιτρεπομένων προσπαθειών του, αλλά η επιλογή πρέπει να αναφέρει ότι αυτό ιδιαίτερα το γράμμα μαντεύτηκε προηγούμενα και μετά κάνει ξανά ανακύκλωση για καινούργιο εισαγόμενο. Στο πρόγραμμα 5.5(a), η F1 αρχικά μηδενίζεται και ορίζεται στο ένα μόνο αν ο εισαγόμενος χαρακτήρας G$ βρέθηκε στη X$.

Η τιμή, X$, πήρε την τιμή "", δηλαδή μία κενή αλφαριθμητική με δείκτη στη γραμμή της αρχικής διαδικασίας, στο πρόγραμμα 5.1(b). Καθώς έγινε μια υπόθεση προστίθεται αυτή στην αλφαριθμητική (ενότητα 8) και έτσι στο στάδιο αυτό είναι αναγκαίο μόνο να διαβάσετε μέσα από την αλφαριθμητική για να ελέγξετε αν κανένα από τα γράμματα είναι ίσο με GUESS$, την τελευταία υπόθεση. Μια μικρή δυσκολία υπάρχει, στο ότι η αλφαριθμητική παίρνει ένα γράμμα παραπάνω κάθε φορά που το γράμμα δεν μαντεύτηκε προηγούμενα, έτσι είναι συνήθως αναγκαίο να ξαναυπολογίσετε το μήκος του (L2) όπως στη γραμμή 5010.

ΠΡΟΓΡΑΜΜΑ 5.5(a)

```
5000 F1=0
5010 L2=LEN(X$)
5020 FOR X=1 TO L2
5030 IF G$=MID$(X$,X,1) THEN F1=-1
5040 NEXT X
5050 IF F1=0 THEN X$=X$+G$
5060 RETURN
```

Ξανά, τρέχοντας το πρόγραμμα δεν θ' αποδώσει καμμία καινούρια εκτύπωση. Ένας άλλος τρόπος ελέγχου του Χ$ είναι με τη χρήση της συνάρτησης INSTR.

5.3.6.1 INSTR

Η συνάρτηση αυτή φάχνει αυτόματα μια αλφαριθμητική με δείκτη από την ύπαρξη ενός δευτέρου. Έτσι οι γραμμές 5010 έως 5050 συμπεριλαμβανομένης θα μπορούσαν ν' αντικατασταθούν με

```
5010 IF INSTR(X$,G$)=0 THEN X$=X$+G$:ELSE F1=-1
```

Η INSTR επιστρέφει ένα μηδέν αν δε βρει την αλφαριθμητική G$ μέσα στο X$. Αν το G$ είναι κάπου στο X$ η INSTR επιστρέφει την θέση του μέσα στο X$. Για παράδειγμα με

```
PRINT INSTR("FRED","R")
```

θα μπορούσατε να πάρετε ένα "2" τυπωμένο. Γιατί το R είναι το δεύτερο γράμμα του "FRED". Εκτός αυτό μπορείτε ακόμα να το κάνετε αυτό με:

```
PRINT INSTR("FRED","ED")
```

το οποίο θα επιστρέφει "3", διότι το "ED" ξεκινάει από τη θέση 3 στο "FRED".

5.3.7 Ενότητα 6: Μήνυμα προηγούμενα δοκιμασμένος χαρακτήρας

Ο σκοπός του μηνύματος αυτού είναι καθαρά να πληροφορήσει τον παίκτη ότι το γράμμα που μόλις μαντεύθηκε έχει ήδη δοκιμαστεί και μετά να καθαρίσει την οθόνη επιστρέφοντας στην προηγούμενη κατάστασή της. Το μήνυμα τυπώνεται (PRINT) πάνω σε μια γραμμή που είναι άδεια. Μόλις εμφανιστεί το μήνυμα στην οθόνη κρατιέται για ένα μικρό διάστημα να το δει ο παίκτης και μετά καθαρίζεται. Αυτό που χρειάζεται εδώ είναι μία τεχνική, που να κάνει το πρόγραμμα να περιμένει μια συγκεκριμένη χρονική περίοδο, δηλαδή καθυστέρηση.

Μόλις απορροφηθεί το μήνυμα στην ενότητα 6 χρειάζεται να μετακινηθεί τυπώνοντας πάνω σ' αυτό κενά διαστήματα. Αυτό πετυχαίνεται μέσω της εντολής SPACE$.

ΠΡΟΓΡΑΜΜΑ 5.6(a)

```
6000 REM character previously tried!
6010 LOCATE 4,24:PEN 2
6020 PRINT"You have already tried that letter!"
6030 FOR x=1 TO 1000:NEXT x
6040 LOCATE 4,24:PEN 1
6050 PRINT SPACE$(36)
6060 RETURN
```

Στο στάδιο αυτό, όταν το πρόγραμμα εκτελεστεί RUN (σε κεφαλαία γράμματα) και ένας χαρακτήρας μαντεύεται για δεύτερη φορά, θα εμφανιστεί το μήνυμα "ήδη δοκιμασμένος".

5.3.8 Είναι η μάντεψη στη λέξη;

Μόλις έχινε η εικασία, η υπορουτίνα στο πρόγραμμα 5.7(a) χρειάζεται να διαβάσει μέσα από τη λέξη ψάχνοντας για ένα ταίρι για το εισαγόμενο γράμμα. Αν βρεθεί ένα τέτοιο ταίρι, τότε η F2 παίρνει την τιμή -1. Προσέξτε ότι, στην αρχή αυτής της υπορουτίνας, το flag ξανατοποθετείται στη τιμή μηδέν και παραμένει μηδέν εκτός αν είναι θετική ή δοκιμή στη γραμμή 7020.

ΠΡΟΓΡΑΜΜΑ 5.7(a)

```
7000 REM IS GUESS IN WORD?
7010 F2=0
7020 IF INSTR(A$,G$)<>0 THEN F2=-1
7030 RETURN
```

Θυμηθείτε ότι ο WORD$() κρατάει την κατάσταση της λέξης που έχει μαντευθεί αρχίζοντας με όλα τα στίγματα. Καθώς μαντεύουμε σωστά τα σωστά γράμματα εισάγονται σ' αυτήν την αλφαριθμητική στην κατάλληλη θέση έτσι η λέξη που πρέπει να μαντευθεί να δημιουργείται προοδευτικά.

Καθώς δεν προσθέθηκε καμμία άλλη εκτύπωση, με το RUN του προγράμματος σ' αυτό το στάδιο δεν θα πάρουμε καινούργια εκτύπωση.

5.3.9 Ενότητα 8: Εκσυγχρονισμός της αλφαριθμητικής με δείκτη των μαντεμένων χαρακτήρων

Η ενότητα αυτή χειρίζεται την περίπτωση όπου ο μαντεμένος χαρακτήρας δεν μαντεύθηκε προηγούμενα. Εισάγεται στο WORD$ στη κατάλληλη θέση.

ΠΡΟΓΡΑΜΜΑ 5.8(a)

```
8000 REM update array
8010 FOR x=1 TO 1
8020 IF g$=MID$(a$,x,1) THEN GOSUB 8100
8030 NEXT x
8040 RETURN
8100 word$(x)=g$
8110 RETURN
```

Το σχήμα 5.18 δείχνει τη διαδικασία για την είσοδο (INPUT) ενός "S" (δηλαδή GUESS$="S") όπου A$="AMSTRAD" και το "S" δεν μαντεύθηκε προηγούμενα.

LOOP NUMBER	A$	WORD$() before	WORD$() after
1	A	.	.
2	M	.	.
3	S	.	S
4	T	.	.
5	R	.	.
6	A	.	.
7	D	.	.

Εικόνα 5.18

5.3.10 Ενότητα 9: Η λέξη ως εδώ

Στο σημείο αυτό, η αλφαριθμητική με δείκτη WORD$ χρειάζεται να συμπληρωθεί από κάθε μια από τις τιμές του WORD$() και μετά το πρόγραμμα θα τυπώσει την τωρινή κατάσταση της μαντεμένης λέξης, αυτό γίνεται στη γραμμή 9060 του 5.9(a) προγράμματος.

ΠΡΟΓΡΑΜΜΑ 5.9(a)

```
9000 REM the word so far
9010 word$=""
9020 FOR x=1 TO 1
9030 word$=word$+word$(x)
9040 NEXT x
9050 LOCATE 16,3:PEN 2
9060 PRINT word$:PEN 1
9070 RETURN
```

5.3.11 Ενότητα 10: Μαντεύθηκαν όλοι οι χαρακτήρες

Για να το ελέγξετε αυτό, ο πίνακας WORD$() πρέπει να διαβαστεί για να δείτε αν καμμία θέση χαρακτήρα παραμένει κενή. Αν συμβαίνει αυτό τότε η F3 παίρνει την τιμή −1 στη γραμμή 10030 του 5.10(a) προγράμματος.

ΠΡΟΓΡΑΜΜΑ 5.10(a)

```
10000 REM ARE ALL CHARACTERS GUESSED?
10010 F3=0
10020 IF WORD$(X)="." THEN F3=-1
10040 NEXT X
10050 RETURN
```

5.3.12 Ενότητα 11: Μήνυμα τα κατάφερες

Η ενότητα αυτή λέει στον παίκτη ότι μαντεύθηκε σωστά η λέξη. Μόλις εμφανιστεί το μήνυμα στην οθόνη το παιχνίδι τελειώνει και δε χρειάζεται επομένως να εμφανιστεί στην οθόνη για ένα συγκεκριμένο χρόνο. Καθώς το επόμενο στάδιο του προγράμματος είναι να ρωτήσει τον παίκτη αν θέλει να ξαναπαίξει, το μήνυμα μπορεί να παραμείνει στην οθόνη μέχρι να πατηθεί κάποιο πλήκτρο.

ΠΡΟΓΡΑΜΜΑ 5.11(a)

```
11000 REM WELL DONE YOU HAVE WON
11010 LOCATE 28,10:PEN 2
11020 PRINT"WELL DONE!"
11030 LOCATE 28,12:PEN 3
11040 PRINT "YOU HAVE WON"
11050 RETURN
```

5.3.13 Ενότητα 12: Θέλετε να ξαναπαίξετε;

Αυτό είναι μία απλή δοκιμή για να δούμε αν χρειάζεται άλλος γύρος. Αν ναι τότε η F5 παίρνει την τιμή -1 (γραμμή 12050) ενώ αν όχι τότε F5=0 (γραμμή 12060).

ΠΡΟΓΡΑΜΜΑ 5.12(a)

```
12000 REM DO YOU WANT ANOTHER GO?
12010 LOCATE 4,24:PEN 2
12020 PRINT"DO YOU WANT ANOTHER GO (Y/N)?"
12030 Z$=INKEY$:IF Z$="" THEN 12030
12040 Z$=UPPER$(Z$)
12050 IF Z$="Y" THEN F5=-1
12060 IF Z$="N" THEN F5=0
12070 IF Z$<>"Y" AND Z$<>"N" THEN 12030
12080 RETURN
```

5.3.14 Ενότητα 13: Αύξηση του μετρητή των λανθασμένων μαντέψεων

Η μεταβλητή "E" καταγράφει τον αριθμό των λανθασμένων μαντειών και απλά αυξάνεται την κατάλληλη ώρα κάθε φορά που η F2 παίρνει την τιμή "0". Έγινε στη γραμμή 13010 του 5.13(a) προγράμματος.

Όταν η μαντεία είναι λανθασμένη, ο παίκτης θα δει το "SORRY THAT LETTER IS NOT IN THE WORD" (το γράμμα αυτό δεν είναι στη λέξη).

ΠΡΟΓΡΑΜΜΑ 5.13(a)

```
13000 REM wrong guess!
13010 e=e+1
13020 LOCATE 4,23:PEN 2
13030 PRINT"I'm sorry that letter is not
 in the word"
13040 FOR x=1 TO 1000:NEXT x
13050 LOCATE 4,23:PEN 1
13060 PRINT SPACE$(40)
13070 RETURN
```

Όπως σε μία προηγούμενη υπορουτίνα το μήνυμα κρατιέται στην οθόνη από την FOR...NEXT στη γραμμή 13040.

5.3.15 Ενότητα 14: Σχεδιασμός κρεμασμένου

Μόλις γίνει μια λανθασμένη εικασία ένα άλλο κομμάτι του κρεμασμένου θα σχεδιαστεί. Ο κρεμασμένος που θα ζωγραφίσουμε, δημιουργείται χρησιμοποιώντας εντολές για graphics (σχεδιασμού) του Amstrad και μια καινούρια εντολή της BASIC.

5.3.15.1 ON...GOTO

Τα ξεχωριστά κομμάτια του κρεμασμένου θα σχεδιαστούν από κομμάτια που βρίσκονται όλα μέσα στη μια υπορουτίνα. Ένα τμήμα θα υπάρξει για να σχεδιάσει τη βάση, ένα για να σχεδιάσει το κεφάλι κ.λ.π. Τα ξεχωριστά αυτά τμήματα θα μπορέσουν να γίνουν βατά ανάλογα με την τιμή της "E" (του μετρητή των λανθασμένων εικασιών), χρησιμοποιώντας μια ειδική έκδοση της εντολής IF...THEN που ονομάζεται ON...GOTO. Αυτή δουλεύει λίγο σαν να μπορούσαν όλες οι εντολές IF...THEN. Παίρνοντας το παράδειγμα:

```
ON X GOTO 100,200,300
```

Ο υπολογιστής καταλαβαίνει σαν:

```
" ON the value of X, GOTO 100, 200, 300"
```

Έτσι αν X=1 τότε το πρόγραμμα κατευθύνεται στη γραμμή 100. Αν X=2 στη γραμμή 200 κ.ο.κ. Αν το X=0 ή μεγαλύτερο από τον αριθμό των γραμμών των ονομάτων που

βρίσκονται μέσα στο πρόγραμμα θα συνεχίσει στην επόμενη γραμμή - στο παρακάτω παράδειγμα, στη γραμμή 30.

Για να το δείτε σε εφαρμογή δοκιμάστε το ακόλουθο:

ΠΡΟΓΡΑΜΜΑ 5.14(b)

```
10 INPUT X
20 ON X GOTO 40,50,60
30 END
40 PRINT"40"
50 PRINT"50"
60 PRINT"60"
```

Τώρα κάντε το RUN και δοκιμάστε το με διαφορετικές εισόδους.

Εκτός από το ότι χρησιμοποιεί τις "GOTO", η εντολή ON... μπορεί να χρησιμοποιεί επίσης και τις "GOSUB". Στην περίπτωση αυτή, το πρόγραμμα κατευθύνεται στην υπορουτίνα με τον ίδιο ακριβώς τρόπο που θα κατευθυνόταν κάθε GOSUB.

Για να σχεδιάσουμε τον κρεμασμένο με γραφικά θα χρησιμοποιήσουμε την εντολή ON...GOTO και κάθε "τμήμα σχεδιασμού" θα τελειώνει με RETURN. Έτσι, στην πρώτη τιμή του E (E=1) το πρόγραμμα θα πάει (GOTO) στη γραμμή 14040 και θα σχεδιάσει μέρος του πλαισίου του κρεμασμένου. Η γραμμή 14090 επιστρέφει το πρόγραμμα πίσω στο Μ.Ε.Π. Μ' αυτό τον τρόπο όλα τα δέκα τμήματα του κρεμασμένου μπορούν να σχεδιαστούν στη σειρά, δηλαδή η τρίτη υπόθεση θα σχεδιάσει το τρίτο "κομμάτι".

ΠΡΟΓΡΑΜΜΑ 5.6(s)

```
14000 REM draw hangman on the value of E
14010 REM
14020 ON e GOTO 14040,14100,14150,14210,
14260,14320,14410,14490,14620,14700
14030 REM the hangman drawing sections
14040 REM--------the frame#1--------
14050 PLOT 240,50:DRAW 240,280
14060 PLOT 242,50:DRAW 242,280
14070 PLOT 244,50:DRAW 244,280
14080 PLOT 246,50:DRAW 246,280
14090 RETURN
```

```
14100 REM--------the frame#2--------
14110 PLOT 246,280:DRAW 376,280
14120 PLOT 246,278:DRAW 376,278
14130 PLOT 246,276:DRAW 376,276
14140 RETURN
14150 REM--------the frame#3--------
14160 PLOT 246,200:DRAW 316,274
14170 PLOT 246,204:DRAW 314,274
14180 PLOT 246,208:DRAW 312,274
14190 PLOT 246,210:DRAW 310,274
14200 RETURN
14210 REM---------the frame#4----------
14220 PLOT 150,50:DRAW 350,50
14230 PLOT 150,48:DRAW 350,48
14240 PLOT 150,46:DRAW 350,46
14250 RETURN
14260 REM--------the rope---------
14270 PLOT 373,274:PLOT 375,276
14280 FOR x=1 TO 20 STEP 4
14290 PLOT 371,276-x:DRAW 376,271-x
14300 NEXT x
14310 RETURN
14320 REM--------the head----------
14330 r=30
14340 PLOT 406,224
14350 py=PI/180
14360 FOR a=0 TO 360 STEP 6
14370 x=r*COS(a*py):y=r*SIN(a*py)
14380 DRAW 376+x,224+y
14390 NEXT a
14400 RETURN
14410 REM--------the face----------
14420 LOCATE 23,11:PRINT"o o"
14430 PLOT 366,209:DRAW 386,209
14440 PLOT 368,209:DRAW 360,216
14450 PLOT 366,209:DRAW 358,216
14460 PLOT 386,209:DRAW 394,216
14470 PLOT 388,209:DRAW 396,216
14480 RETURN
14490 REM--------the body----------
14500 PLOT 348,194:DRAW 408,194
14510 DRAW 420,160:DRAW 400,130
14520 PLOT 348,194
```

```
14530 DRAW 336,160:DRAW 354,130
14540 PLOT 356,180
14550 DRAW 349,160:DRAW 363,135
14570 PLOT 398,180
14580 DRAW 406,160:DRAW 392,135
14590 DRAW 400,130
14600 PLOT 363,135:DRAW 350,130
14610 RETURN
14620 REM--------the legs----------
14630 PLOT 355,135
14640 DRAW 363,80:DRAW 375,80
14650 PLOT 410,143:DRAW 402,80
14660 DRAW 390,80:DRAW 386,145
14670 PLOT 375,80:DRAW 378,145
14680 PLOT 355,145:DRAW 395,145
14690 RETURN
14700 REM--------the shoes----------
14710 PLOT 363,80:DRAW 345,70
14720 DRAW 375,70:DRAW 375,80
14730 PLOT 404,80:DRAW 422,70
14740 DRAW 390,70:DRAW 390,80
14750 RETURN
```

5.3.16 Ενότητα 15: Χρησιμοποιήθηκαν όλοι οι γύροι;

Η ενότητα αυτή είναι μια απλή δοκιμή της μεταβλητής "E". Αν χρησιμοποιήθηκαν όλοι οι επιτρεπόμενοι γύροι τότε η E έχει την τιμή 10 και η F4 την −1.

ΠΡΟΓΡΑΜΜΑ 5.15(a)

```
15000 REM ALL GOES USED UP?
15010 F4=0
15020 IF E=10 THEN F4=-1
15030 RETURN
```

5.3.17 Ενότητα 16: Εμφάνιση του μηνύματος της ήτας

Στο σημείο αυτό στο παιχνίδι, όλα τελείωσαν για τον παίκτη και αυτό ακριβώς λέει το μήνυμα στις γραμμές 16000 και 16080. Δίνεται στον παίκτη η σωστή λύση σαν μία αποζημίωση.

ΠΡΟΓΡΑΜΜΑ 5.16(a)

```
16000 REM LOSER MESSAGE
16010 LOCATE 2,10:PEN 2
16020 PRINT"BAD LUCK!"
16030 LOCATE 2,12
16040 PRINT"YOU ARE HUNG"
16050 LOCATE 2,14:PEN 3
16060 PRINT"THE WORD WAS"
16070 LOCATE 2,16:PRINT A$
16080 RETURN
```

Τώρα που έχουν γραφεί οι ρουτίνες και το παιχνίδι δουλεύει θα μπορούσατε να αναρωτηθείτε γιατί τα flags ορίστηκαν "−1" αντί για το συνήθες "1". Όταν τα flags ορίζονται έτσι μπορούν να δοκιμαστούν "λογικά" για να δούμε αν είναι αληθή ή ψευδή.

5.4 Αληθές/Ψευδές

Ο Amstrad έχει δημιουργήσει ειδικές λογικές ρουτίνες. Μια μεταβλητή με την τιμή −1 θεωρείται ότι είναι ΑΛΗΘΗΣ. Αυτό σημαίνει ότι έχει συναντηθεί η συνθήκη που δηλώνει το flag. Αντί να πείτε:

```
IF F1=-1 THEN GOSUB 6000:GOTO 530
```

Χρησιμοποιώντας τη λογική του Amstrad στο χειρισμό της γραμμής μπορεί να ξαναγραφεί.

```
IF F1 THEN GOSUB 6000:GOTO 530
```

Ο Amstrad το μεταφράζει ως εξής:

> "Αν η μεταβλητή F1 δηλώνει μια αληθινή τιμή δηλαδή ισούται με "−1", τότε εκτελεί το υπόλοιπο της γραμμής"

Παραλείποντας το "−1" ο υπολογιστής εκτελεί πολύ γρηγορότερα την εργασία ελέγχου, σχεδόν δύο φορές γρηγορότερα. Βέβαια συναλλασσόμαστε με χιλιοστά του δευτερολέπτου, αλλά όπως λένε "ένα εκατομμύριο μισές

πέννες είναι πάρα πολλές μισές πέννες".

Η εντολή **NOT** χρησιμοποιείται από τον Amstrad για να καθορίσει αν η μεταβλητή έχει την τιμή μηδέν δηλαδή την κατάσταση όχι αληθή! (Ψευδή). Η γραμμή 670 του Μ.Ε.Π. μπορεί να μετασχηματιστεί από:

```
670 IF F4 =0 THEN 530
```

το:

```
670 IF NOT F4 THEN 530
```

Η εργασία με NOT, όπως η αληθής, είναι γρηγορότερη από μια κανονική IF...THEN λειτουργία. Χρησιμοποιώντας NOT και το TRUE, οποτεδήποτε μπορείτε να δοκιμάσετε τιμές, μπορεί να αυξήσει την ταχύτητα των προγραμμάτων, ειδικά σε μεγάλα προγράμματα με πολλά flags.

ПРОГРАММА 5

```
500 GOSUB 1000:REM initialisation
510 GOSUB 2000:REM choose word
520 GOSUB 3000:REM display screen
530 GOSUB 4000:REM input guess
540 GOSUB 5000:REM character previously
tried?
550 IF f1 THEN GOSUB 6000:GOTO 530:REM g
o back for another input
560 GOSUB 7000:REM guess in word?
570 IF NOT f2 THEN 640:REM guess not in
word
580 GOSUB 8000:REM update array
590 GOSUB 9000:REM display screen
600 GOSUB 10000:REM any characters left
to guess?
610 IF f3 THEN 530
620 GOSUB 11000:REM well done
630 GOTO 700:REM another go?
640 GOSUB 13000:REM increment wrong gues
s count
650 GOSUB 14000:REM display new screen
660 GOSUB 15000:REM are all goes used?
670 IF NOT f4 THEN 530:REM more guesses
680 GOSUB 16000:REM you have lost
700 GOSUB 12000:REM another go?
710 IF f5 THEN 500:REM play it again Sam
!
720 CLS:LOCATE 10,10
730 PRINT"Goodbye!"
740 END
899 END
900 LOCATE 4,20:PRINT"Press any key to c
ontinue"
910 a$=INKEY$:IF a$="" THEN 910:ELSE RET
URN

1000 REM initialisation
1010 MODE 1
1020 INK 0,13:INK 1,26:INK 3,17:INK 4,3
1030 BORDER 17:PEN 1:PAPER 0:CLS
1040 LOCATE 14,2:PRINT"***HANGMAN***"
1050 LOCATE 4,6
1060 PRINT"Here are the rules!"
1070 LOCATE 4,8
1080 PRINT"There are no rules!"
1190 LOCATE 4,20
1200 PRINT"Press space bar to continue"
```

```
1205 IF INKEY$<>" " THEN 1190
1210 z$=" first "
1220 e=0
1230 x$=""
1240 RETURN
2000 REM choose word
2010 r=INT(RND*10)+1
2011 r=4:REM development only
2020 RESTORE
2030 FOR x=1 TO r
2040 READ a$
2050 NEXT x
2060 l=LEN(a$)
2500 word$=""
2510 FOR x=1 TO l
2520 word$(x)=".":word$=word$+"."
2530 NEXT x
2540 RETURN
3000 REM display the screen
3005 CLS
3010 LOCATE 14,1:PRINT"***HANGMAN***"
3020 LOCATE 4,3
3030 PRINT"The word is"
3040 LOCATE 16,3:PEN 2
3050 PRINT word$
3060 LOCATE 4,5:PEN 1
3070 PRINT"The word has";
3080 PEN 2:PRINT l;:PEN 1
3090 PRINT"letters"
3100 RETURN
4000 REM input a guess
4010 LOCATE 4,7
4020 PRINT"What is your";z$;"guess?    "
4025 LOCATE 29,7:PEN 3:PRINT CHR$(143):P
EN 1
4030 g$=INKEY$:IF g$="" THEN 4030
4040 g$=UPPER$(g$)
4050 IF g$<"A" OR g$>"Z" THEN 4030
4060 LOCATE 29,7:PEN 2:PRINT g$
4070 z$=" next ":PEN 1
4080 RETURN
5000 f1=0
5010 IF INSTR(x$,g$)=0 THEN x$=x$+g$:ELS
E f1=-1
5060 RETURN
6000 REM character previously tried!
6010 LOCATE 4,24:PEN 2
6020 PRINT"You have already tried that l
etter!"
6030 FOR x=1 TO 1000:NEXT x
```

```
6040 LOCATE 4,24:PEN 1
6050 PRINT SPACE$(36)
6060 RETURN
7000 REM is guess in word?
7010 f2=0
7020 IF INSTR(a$,g$)<>0 THEN f2=-1
7030 RETURN
8000 REM update array
8010 FOR x=1 TO l
8020 IF g$=MID$(a$,x,1) THEN GOSUB 8100
8030 NEXT x
8040 RETURN
8100 word$(x)=g$
8110 RETURN
9000 REM the word so far
9010 word$=""
9020 FOR x=1 TO l
9030 word$=word$+word$(x)
9040 NEXT x
9050 LOCATE 16,3:PEN 2
9060 PRINT word$:PEN 1
9070 RETURN
10000 REM are all characters guessed?
10010 f3=0
10020 FOR x=1 TO l
10030 IF word$(x)="." THEN f3=-1
10040 NEXT x
10050 RETURN
11000 REM well done you have won
11010 LOCATE 28,10:PEN 2
11020 PRINT"Well done!"
11030 LOCATE 28,12:PEN 3
11040 PRINT"You have won!
11050 RETURN
12000 REM do you want another go?
12010 LOCATE 4,24:PEN 2
12020 PRINT"Do you want another go (Y/N)
?"
12030 z$=INKEY$:IF z$="" THEN 12030
12040 z$=UPPER$(z$)
12050 IF z$="Y" THEN f5=-1
12060 IF z$="N" THEN f5=0
12070 IF z$<>"Y" AND z$<>"N" THEN 12030
12080 RETURN
13000 REM wrong guess!
13010 e=e+1
13020 LOCATE 4,23:PEN 2
13030 PRINT"I'm sorry that letter is not
 in the word"
```

```
13040 FOR x=1 TO 1000:NEXT x
13050 LOCATE 4,23:PEN 1
13060 PRINT SPACE$(40)
13070 RETURN
14000 REM draw hangman on the value of E
14010 REM
14020 ON e GOTO 14040,14100,14150,14210,
14260,14320,14410,14490,14620,14700
14030 REM the hangman drawing sections
14040 REM--------the frame#1--------
14050 PLOT 240,50:DRAW 240,280
14060 PLOT 242,50:DRAW 242,280
14070 PLOT 244,50:DRAW 244,280
14080 PLOT 246,50:DRAW 246,280
14090 RETURN
14100 REM--------the frame#2--------
14110 PLOT 246,280:DRAW 376,280
14120 PLOT 246,278:DRAW 376,278
14130 PLOT 246,276:DRAW 376,276
14140 RETURN
14150 REM--------the frame#3--------
14160 PLOT 246,200:DRAW 316,274
14170 PLOT 246,204:DRAW 314,274
14180 PLOT 246,208:DRAW 312,274
14190 PLOT 246,210:DRAW 310,274
14200 RETURN
14210 REM--------the frame#4----------
14220 PLOT 150,50:DRAW 350,50
14230 PLOT 150,48:DRAW 350,48
14240 PLOT 150,46:DRAW 350,46
14250 RETURN
14260 REM--------the rope---------
14270 PLOT 373,274:PLOT 375,276
14280 FOR x=1 TO 20 STEP 4
14290 PLOT 371,276-x:DRAW 376,271-x
14300 NEXT x
14310 RETURN
14320 REM--------the head-----------
14330 r=30
14340 PLOT 406,224
14350 py=PI/180
14360 FOR a=0 TO 360 STEP 6
14370 x=r*COS(a*py):y=r*SIN(a*py)
14380 DRAW 376+x,224+y
14390 NEXT a
14400 RETURN
14410 REM--------the face-----------
14420 LOCATE 23,11:PRINT"o o"
14430 PLOT 366,209:DRAW 386,209
14440 PLOT 368,209:DRAW 360,216
```

```
14450 PLOT 366,209:DRAW 358,216
14460 PLOT 386,209:DRAW 394,216
14470 PLOT 388,209:DRAW 396,216
14480 RETURN
14490 REM---------the body-----------
14500 PLOT 348,194:DRAW 408,194
14510 DRAW 420,160:DRAW 400,130
14520 PLOT 348,194
14530 DRAW 336,160:DRAW 354,130
14540 PLOT 356,180
14550 DRAW 349,160:DRAW 363,135
14570 PLOT 398,180
14580 DRAW 406,160:DRAW 392,135
14590 DRAW 400,130
14600 PLOT 363,135:DRAW 350,130
14610 RETURN
14620 REM---------the legs-----------
14630 PLOT 355,135
14640 DRAW 363,80:DRAW 375,80
14650 PLOT 410,143:DRAW 402,80
14660 DRAW 390,80:DRAW 386,145
14670 PLOT 375,80:DRAW 378,145
14680 PLOT 355,145:DRAW 395,145
14690 RETURN
14700 REM---------the shoes-----------
14710 PLOT 363,80:DRAW 345,70
14720 DRAW 375,70:DRAW 375,80
14730 PLOT 404,80:DRAW 422,70
14740 DRAW 390,70:DRAW 390,80
14750 RETURN
15000 REM all goes used up?
15010 f4=0
15020 IF e=10 THEN f4=-1
15030 RETURN
15040 IF a$="N" THEN f4=0
15050 IF a$<>"Y" AND a$<>"N" THEN 15020:
ELSE RETURN
16000 REM loser message
16010 LOCATE 2,10:PEN 2
16020 PRINT"Bad luck!"
16030 LOCATE 2,12
16040 PRINT"You are hung!"
16050 LOCATE 2,14:PEN 3
16060 PRINT"The word was"
16070 LOCATE 2,16:PRINT a$
16080 RETURN
18000 DATA CAT,TURBINE,PLATE,COMPUTER
18010 DATA
18020 DATA
18030 DATA
```

5.5 Βελτιώσεις του προγράμματος

'Ενας από τους ανεπίσημους κανόνες σε μια εταιρεία υπολογιστών είναι ότι ο προγραμματιστής ποτέ δεν τελειώνει ένα πρόγραμμα. Το πρόγραμμα μπορεί ν' αναπτυχθεί μέχρι το στάδιο που θα μπορεί να παιχτεί (το παιχνίδι κρεμάλα είναι σ' αυτό το στάδιο) αλλά είναι δυνατές πάντα κάποιες βελτιώσεις. 'Οταν ένα πρόγραμμα έχει γραφτεί με δομημένο τρόπο είναι φυσιολογικά πολύ εύκολο να εισάγουμε μερικά "παραπέρα πράγματα" για να κάνουμε ένα κατά τα άλλα καλό παιχνίδι ακόμα καλύτερο. Παρακάτω είναι μερικές προτάσεις πάνω στις οποίες θα σας άρεσε να δουλέψετε. Περισσότερες από τις προτάσεις είναι μια πρώτη γεύση και αφέθηκαν σε σας να τις εφαρμόσετε.

* Καλύτερες εκτυπώσεις στην οθόνη

* Δυσκολότερο επίπεδο λέξεων

* 'Οσο καλύτερος είναι ο παίκτης τόσο λιγότεροι είναι οι επιτρεπόμενοι γύροι

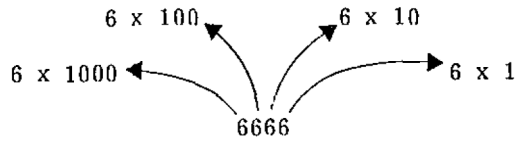
* Παιχνίδι δύο παικτών

1. ΠΑΡΑΡΤΗΜΑ 1

1.1 Δυαδικός, Κωδικοποιημένος-Δυαδικός, Δεκαδικός και Δεκαεξαδικός Συμβολισμός

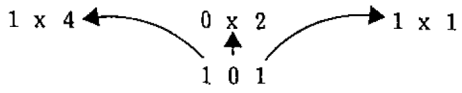
Τα συστήματα μέτρησης γενικά χρησιμοποιούν σ' όλο τον κόσμο το δεκαδικό σύστημα και αυτό αναπτύχθηκε για να μετρά και πάνω από το 10 και κάτω από το 1. Ψηφία αριστερά του αριθμού είναι μεγαλύτερης τιμής από ότι τα δεξιά, για παράδειγμα στον αριθμό 66 το πρώτο 6 έχει μια τιμή 10πλάσια του δευτέρου δηλαδή

Αυτό επεκτείνεται σε μεγαλύτερους αριθμούς όπου τα αριστερά ψηφία είναι διαδοχικά μεγαλύτερα κατά το δεκαπλάσιο δηλαδή



'Ενα σύστημα όπου η θέση ή ο χώρος ενός ψηφίου σ' έναν αριθμό επηρεάζει την τιμή του, είναι γνωστό σαν ΘΕΣΗ-ΤΙΜΗ αριθμητικό σύστημα. Στο δεκαδικό σύστημα, οι τιμές των ψηφίων αυξάνονται κατά πολλαπλάσιο του 10 και αυτό είναι γνωστό σαν η ΒΑΣΗ γι αυτό το σύστημα. 'Αλλα συστήματα χρησιμοποιούν άλλες βάσεις, αλλά ακολουθούν την ίδια λογική όπως το δεκαδικό σύστημα δηλαδή ο χώρος αριστερά είναι μεγαλύτερος κατά πολλαπλάσιο της βάσης.

Ο υπολογιστής, που είναι βασικά ηλεκτρικός στη λειτουργία του, μπορεί να αναγνωρίσει μόνο δύο καταστάσεις, την ανοικτή και την κλειστή ("0" και "1" αντίστοιχα) και γι αυτό χρησιμοποιεί το Δυαδικό σύστημα - βάση 2. 'Ετσι κάθε αριθμός στο δυαδικό αποτελείται απλά από μηδέν και άσσους ή μιλώντας ηλεκτρικά, με κλεισίματα και ανοίγματα (ή ηλεκτρονικά, μηδέν volts -OFF- και μερικά volts -ON-). Για να μετρήσει μετά το ένα, το δυαδικό σύστημα πρέπει να καταφύγει σε θέση - τιμή συμβολισμό και όπως σ' άλλες περιπτώσεις ο πολλαπλασιαστικός παράγοντας είναι η βάση δηλαδή 2. 'Ετσι ο αριθμός 101 με βάση το 2 ή δυαδικά παρουσιάζει:



δηλαδή 4+0+1=5. Προφανώς η πληθώρα των βάσεων μας δημιουργεί πρόβλημα όταν απεικονίζουμε τούς αριθμούς, καθώς με βάση το 10 το "101" παριστάνει το 101, ενώ στο δυαδικό (βάση 2) το "101" παριστάνει το 5. Για να παρακάμψουμε αυτή την αμφιβολία, υπάρχει ένας τύπος όταν απεικονίζουμε αριθμούς στον οποίο η βάση γράφεται στα δεξιά του αριθμού και ακριβώς κάτω από τη γραμμή. Έτσι οι δύο αριθμοί που συζητήθηκαν παραπάνω γίνονται:

$$101_{10} = 101 \text{ με βάση το } 10$$
$$101_{2} = 5 \text{ με βάση το } 2$$

Η σύγχρονη γενιά των home-computers (μοντέλα 1985) χρησιμοποιούν καταχωρητές ή μνήμες 8 bit και μπορούν έτσι να απεικονίσουν αριθμούς μέχρι 11111111 , δηλαδή στη βάση 10:

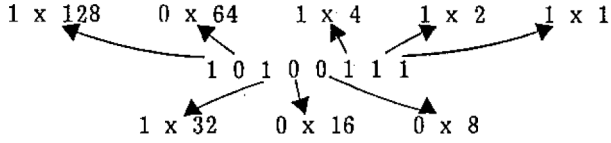
128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 $= 255_{10}$

1	1	1	1	1	1	1	1	Digit
128	64	32	16	8	4	2	1	Equivalent in base 10

Εικόνα A1.1

Και με παράδειγμα, ας πάρουμε μια ακόμα μετατροπή - έστω, **10100111** .

By way of example, let's take one more conversion - say, 10100111_2



Thus 10100111 = 1x128 +0x64 +1x32 +0x16+ 0x8 +1x4 +1x2 +1x1
= 128+32+4+2+1
= 167_{10}

Μόνο για να ελέγξετε ότι έχεις καταλάβει δοκίμασε το ακόλουθο:

ΑΣΚΗΣΗ Α1.1

Υπολογίστε την τιμή των παρακάτω με βάση το 10.

i. 00000011

ii. 00000100

iii. 10000000

iv. 10000011

v. 10110111

vi. 01110011

Οι απαντήσεις δίνονται στο κεφάλαιο των λύσεων.

1.1.1 Δεκαεξαδικός

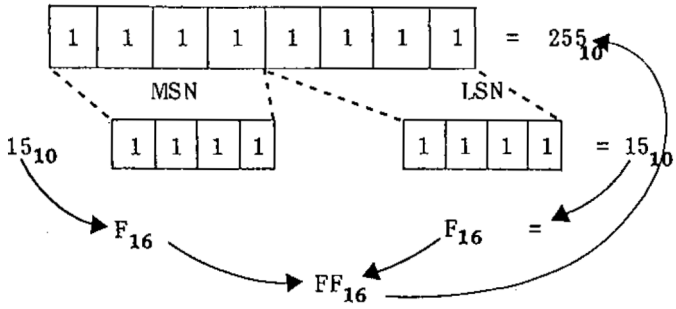
Ενώ τα μηδέν και ένα είναι βολικά για τον υπολογιστή, είναι πολύ λιγότερο για τον απλό άνθρωπο, έτσι αναζητήθηκε ένας συμβιβαστικός τρόπος. Ο δεκαδικός συμβολισμός είναι μικρής χρήσης καθώς μακρυά από το 1 και 1 δεν υπάρχει άλλη αντιστοιχία. Μια παρεταίρω ιδέα θα μπορούσε να είναι να πάρουμε όλα τα 8 δυαδικά bits σαν ένα ψηφίο (δηλαδή μέχρι 255) και να *χρησιμοποιήσουμε μια βάση την 256. Τι το απαράδεκτο* θα μπορούσες να δείτε σ' αυτό; Αυτό μακριά από την ιδέα μόνο του είναι λίγο ακατανόητο! Ώρα για σκέψη. Η απάντηση έρχεται από την εξέταση της περίπτωσης της βάσης 10 στην οποία χρειάζονται 10 ψηφία (0-9) έτσι η βάση 256 θα χρειαζόταν 256 ψηφία.

Ένα αποδεκτά συμβιβαστικό σύστημα χώρισε τα 8 bits σε δύο μέρη και τα απεικονίζει αυτά ξεχωριστά. Έτσι, ο μεγαλύτερος αριθμός που πρέπει να παρασταθεί είναι ο 1111 ή 15 και αυτά με τη βοήθεια του 0, χρειάζονται 16 διαφορετικά σύμβολα. Αυτά που χρησιμοποιήθηκαν για αυτή τη δουλειά είναι:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Decimal number
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Symbol

Εικόνα A1.2

Χρησιμοποιώντας αυτόν τον συμβολισμό, κάθε αριθμός 8 bit μπορεί να απεικονιστεί με 2 σύμβολα, ένα για τα πιο σημαντικά 4 bits και ένα για τα λιγότερα σημαντικά 4 bits. Για να αποφύγουμε τη μάλλον μακρά περιγραφή των δύο αυτών μισών ενός byte, τους δόθηκε ο όρος NYBBLES. Έτσι το byte αποτελείται από δύο nybbles το πιο σημαντικό (MSN) και το λιγότερο σημαντικό nybble (LSN) – δείτε το σχήμα A1.3



Εικόνα A1.3

Το σύστημα που περιγράφτηκε και χρησιμοποιεί 16 σύμβολα, του δόθηκε το όνομα Δεκαεξαδικό – συνήθως συγκοπτόμενα HEX. Το σημαντικότερο του πλεονέκτημα, όσον αφορά τους υπολογιστές, είναι η συμβατότης του με το δυαδικό σύστημα. Κάθε δυαδικός αριθμός 8 bits μπορεί να παρασταθεί με δύο 16δικούς χαρακτήρες.

Με σκοπό να αποκτήσετε μια εμπειρία των hex δοκιμάστε το ακόλουθο:

ΑΣΚΗΣΗ Α1.2

Υπολογίστε την τιμή σε δεκαδικό των παρακάτω:

i. 0009

ii. 0013

iii. 00A5

iv. 0AAE

v. 000E

vi. 011A

vii. 00EA

Οι απαντήσεις δίνονται στο κεφάλαιο των λύσεων.

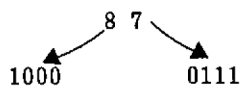
Αν χρειάζεστε περισσότερες ασκήσεις στους 16δικούς το Bin/Hex πρόγραμμα ασκήσεων θα σας τις προμηθεύσει, η επιλογή "1" για δεκαδικό σε 16δικό και "4" για 16δικό σε δεκαδικό. Βυμηθείτε το <DELETE> για να διαγράψετε την τελευταία είσοδο και το <RETURN> για το MENU.

1.1.2 Κωδικοποιημένος - Δυαδικός Δεκαδικός

Με την ίδια επιτυχία που χρησιμοποιούμε στους υπολογισμούς τους δυαδικούς, δεκαδικούς και 16δικούς συμβολισμούς, χρησιμοποιούμε και ένα άλλο σύστημα το Κωδικοποιημένο - Δυαδικό σύστημα (BCD) Binary - Coded Decimal. Όπως αποδεικνύει το όνομά του, είναι μια υβριδική μορφή με στοιχεία συχρόνως και από το δυαδικό και από το δεκαδικό σύστημα. Χρησιμοποιείται συνήθως όπου χρειάζεται μια έξοδος με ψηφιακή μορφή, π.χ. ένα ψηφιακό ρολόϊ, ή όταν χρειάζεται μεγάλη ακρίβεια και δεν επιτρέπεται να πεταχτούν bits.

Στο BCD σύστημα, η φυσιολογική δεκαδική βάση διατηρείται, δηλαδή μία θέση είναι ένας παράγοντας 10πλάσιος της γειτονικής, αλλά κάθε ιδιαίτερο ψηφίο

παριστάνεται σε δυαδικό. Έτσι ο αριθμός 87 θα μπορούσε να παρασταθεί:



i.e. BCD = 1000 0111 (or in eight bits 10000111)

Εικόνα A1.4 (A1.6)

Καθώς το μεγαλύτερο ψηφίο που χρειάζεται στο δεκαδικό συμβολισμό είναι το 9, μόνο 4 δυαδικά bits χρειάζονται να το παραστήσουν, δηλαδή 9 =1001 έτσι ένα BCD ψηφίο μπορεί να παρασταθεί με 1 nybble και 2 ψηφία μ' ένα byte. Το σχήμα A1.6 το δείχνει, όπου το 87 παριστάνεται σε BCD σαν 10000111 Αυτό μπορεί να δημιουργήσει ασάφεια στο ότι ο δυαδικός είναι ο 10000111 Για να το υπερνικήσει, οι παραστάσεις σε BCD θα δίνονται με το συμβολισμό 10000111(BCD).

Χρησιμοποιώντας 4 δυαδικά bits είναι πιθανό να μετρήσει μέχρι το 15 (δηλαδή 1111 =15), αλλά σε BCD το μεγαλύτερο ψηφίο είναι το 9, έτσι αναπόφευκτα το BCD είναι λιγότερο οικονομικό στη χρήση του στο χώρο. Το μεγαλύτερο ψηφίο, το 9 είναι το 1001 και όταν προστεθεί σ' αυτό η μονάδα επιστρέφει στο 0000 και μεταφέρει τη μονάδα στο επόμενο nybble δηλαδή.

```
8    =  0000 1000 (Βάση 2 BCD)
9    =  0000 1001   "   "  "
10   =  0001 0000   "   "  "
11   =  0001 0001   "   "  "
```

Εικόνα A1.5 (A1.7)

Τώρα που ξέρεις τα πάντα για το BCD, δοκίμασε το ακόλουθο:

ΑΣΚΗΣΗ Α1.3

Μετέτρεψε τους ακόλουθους δεκαδικούς αριθμούς σε BCD:

i. 4

ii. 10

iii. 77

iv. 97

v. 53

vi. 102

vii. 953

viii. 2579

Οι απαντήσεις δίνονται στο κεφάλαιο των λύσεων.

ΑΣΚΗΣΗ Α1.4

Μετέτρεψε τους ακόλουθους BCD σε δεκαδικούς.

i. 0000 0001

ii. 0000 1001

iii. 0001 0101

iv. 0010 0000

v. 0100 1001

vi. 1010 0011

vii. 1001 0111

viii. 1000 1000

Οι απαντήσεις δίνονται στο κεφάλαιο των λύσεων.

Στις επεξηγήσεις που δόθηκαν για την τιμή της θέσης στον θέση-τιμή συμβολισμό, υιοθετήθηκε μια απλούστευση, με σκοπό να δώσουμε σαφέστερες εξηγήσεις για τους λιγότερο ταλαντούχους στα μαθηματικά! Ωστόσο, αν επιθυμείτε λίγο περισσότερες μαθηματικές εξηγήσεις δεν έχετε παρά να μελετήσετε. Αλλιώς - τέλος του πρώτου παραρτήματος.

Στους δυαδικούς αριθμούς, λεχόταν ότι οι θέσεις αύξαναν τις τιμές τους σε πολλαπλάσια του 2, αλλά τα λιγότερο σημαντικά bit του δυαδικού αριθμού ήταν ισοδύναμα με τον ίδιο συμβολισμό στη βάση 10 (ή για τον ίδιο λόγο στη βάση 3, ή οτιδήποτε). Στην πραγματικότητα πολλαπλασιαστικός παράγοντας είναι η βάση, υψωμένη στη δύναμη της θέσης του, ξεκινώντας από το μηδέν και προς τ' αριστερά. Δηλαδή δυαδικά:

7	6	5	4	3	2	1	0
128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Place
Previously stated multiplication factor

Mathematically more precise factor.

Έτσι το λιγότερο σημαντικό bit πολλαπλασιάζεται με 2 ή 1. (Αν δεν είσαστε σίγουροι γι αυτό δοκιμάστε το απευθείας πρόγραμμα PRINT 2^10.) Το επόμενο β πολλαπλασιάζεται με 2 κ.ο.κ. Αυτός ο κανόνας ισχύει για ΚΑΘΕ βάση, ας το εφαρμόσουμε για τους 16δικούς δηλαδή βάση 16.

Ο λιγότερο σημαντικός παράγοντας bit = 16 = 1

Ο 2ος σημαντικός παράγοντας bit = 16 = 16

Ο 3ος πιο σημαντικός παράγοντας bit = 16 = 256

Ο πιο σημαντικός παράγοντας bit = 16 = 4096

2. ΠΑΡΑΡΤΗΜΑ 2

2.1 Χρήση του Κασετοφώνου

Το δεύτερο κεφάλαιο περικλείει μία σύντομη περιγραφή των εντολών **SAVE** και **LOAD** δείχνοντας στο χρήστη πως να σώσει ένα πρόγραμμα πάνω στην ταινία και μετά να το ξαναφορτώσει. Οι εντολές που χρησιμοποιούνται είναι οι απλούστερες μορφές των SAVE και LOAD. Αυτό το παράρτημα θα περιγράφει τις εξτρα ιδιότητες του συστήματος με κασέτες για τον Amstrad.

2.1.1 Save

Η εντολή Save μπορεί να χρησιμοποιηθεί για να σώσει τρεις τύπους αρχείων, κάθε τύπος διαλέγεται με κώδικα.

2.1.2 Save ''όνομα αρχείου'',A

'Οταν χρησιμοποιούμε SAVE,A ένα BASIC πρόγραμμα σώζεται σαν ASCII αρχείο. Το πρόγραμμα GUESSER στο κεφάλαιο 2 σώθηκε σαν ASCII αρχείο. Αν δεν επιλέγει ο τύπος του αρχείο τότε το "A" δεν έχει νόημα. Ο Amstrad (όπως οι περισσότεροι computers) σώζουν προγράμματα BASIC σε αριθμητική μορφή. Αυτή η ASCII τιμή, με τη σειρά της, μετατρέπεται σ' έναν ηλεκτρονικό ήχο που αποθηκεύεται πάνω στη ταινία.

2.1.3 Save ''Ονομα αρχείου'',P

Το P σ' αυτή την περίπτωση σημαίνει προστασία. 'Ενα σωσμένο πρόγραμμα μ' αυτόν τον τρόπο δεν μπορεί να φορτωθεί με το φυσιολογικό τρόπο. Για να διευκρινίσετε αυτό το σημείο κάνετε τ' ακόλουθα:

* Πληκτρολογείστε:

```
10 REM PROTECTED PROGRAM
20 PRINT "I AM SAFE"
```

* SAVE "PROTECT",P

* Ο Amstrad θα ανταποκριθεί με PRESS REC and PLAY then any key:

* Πατείστε τα REC και PLAY του κασετοφώνου μαζί και μετά πάτα το SPACE BAR

* Ο υπολογιστής θα εμφανίσει το ακόλουθο μήνυμα Saving PROTECT block 1

* 'Οταν εμφανιστεί το μήνυμα Ready, χώρισε την κασέτα από την αρχή και πληκτρολόγησε: LOAD "PROTECT"

* Ο Amstrad θ' αναφέρει πάλι Press PLAY then any key:

* *Πατείστε το PLAY του κασετοφώνου και μετά πάτα το* SPACE BAR (το πλήκτρο για τα διαστήματα)

* Ο υπολογιστής θα αρχίσει να ψάχνει για το πρόγραμμα που καλείται PROTECT. 'Οταν το βρει θα δεις το ακόλουθο μήνυμα: Loading PROTECT block 1

* 'Οταν εμφανιστεί το μήνυμα Ready ο υπολογιστής έχει τελειώσει το φόρτωμα. 'Ολα φαίνονται φυσιολογικά.

* Πληκτρολογείστε: LIST

* Τίποτα δεν εμφανίζεται. Το πρόγραμμά σας δε φορτώθηκε. Και όχι μόνο αυτό αλλά οτιδήποτε ήταν μέσα στη μνήμη του υπολογιστή πριν το φόρτωμα έχει διαγραφεί.

'Ενα πρόγραμμα που σώζεται χρησιμοποιώντας προστασία μπορεί να ξαναφορτωθεί μόνο μ' έναν από τους δύο τρόπους. Ο πρώτος είναι η εντολή CHAIN.

2.1.3.1 CHAIN

Η εντολή CHAIN είναι μια πολύ ειδική παραλλαγή της εντολής LOAD. 'Οταν χρησιμοποιείται η CHAIN το πρόγραμμα που φορτώνεται τρέχει (RUN) αυτόματα από ένα συγκεκριμένο αριθμό εντολής.

```
CHAIN "CUESSER",100
```

Αυτό θα φορτώσει το BASIC πρόγραμμα GUESSER και θ' αρχίσει να τρέχει από τη γραμμή 100. Αν δεν καθοριστεί γραμμή προγράμματος, τότε το πρόγραμμα θα ξεκινήσει από την πρώτη γραμμή του προγράμματος. Αν ο αριθμός που χρησιμοποιείται στην εντολή CHAIN δεν υπάρχει τότε ο υπολογιστής θ' αναφέρει:

```
Line does not exist
```

και το φόρτωμα που επιχειρήθηκε θάναι ανεπιτυχές.

2.1.3.2 RUN

'Αλλη μια εντολή που χρησιμοποιείται για να φορτώνει προγράμματα που έχουν σωθεί με προστασία είναι η εντολή RUN. 'Οταν χρησιμοποιείται στο φόρτωμα, συνοδεύεται από το όνομα του αρχείου. Π.χ.

```
RUN "PROTECT"
```

Αυτό θα φορτώσει το πρόγραμμα PROTECT και θα τρέξει το πρόγραμμα από τη γραμμή με το μικρότερο αριθμό.

'Εχοντας πατημένο το CTRL, και πιέζοντας το μικρό ENTER στο αριθμητικό πληκτρολόγιο προκαλεί τον Amstrad να εμφανίσει το:

```
RUN"
Press PLAY then any key:
```

Αυτό θα φορτώσει και θα τρέξει το αμέσως επόμενο πρόγραμμα στην ταινία.

Και οι δύο μαζί εντολές CHAIN και RUN μπορούν να χρησιμοποιηθούν με τον ίδιο τρόπο για προστατευμένα και απροστάτευτα προγράμματα.

2.1.3.3 SAVE ''όνομα αρχείου'',B

Αυτή η μορφή του Save χρησιμοποιείται για να σώσει τα περιεχόμενα καθορισμένων θέσεων μνήμης. Τα δεδομένα αποθηκεύονται στην ταινία σαν δυαδικά δεδομένα. Ο χρήστης πρέπει να πληκτρολογήσει τη θέση από όπου πρέπει ν' αρχίσει το σώσιμο και πόσες θέσεις να σώσει. Για παράδειγμα:

```
SAVE"MEMORY",B,55000,100
```

Αυτό θα σώσει τα περιεχόμενα όλων των θέσεων μνήμης από 55000 έως 55100.

Υπάρχει μια άλλη παράμετρος που μπορεί να χρησιμοποιηθεί με SAVE,B και αυτή είναι το σημείο εισόδου. Αν αυτή η τιμή περιέχεται τότε το αρχείο θ' αρχίσει την εκτέλεση από εκείνο το σημείο. Για παράδειγμα:

```
SAVE"MEMORY",B,55000,100,55020
```

Αυτή θα σώσει το ίδιο block μνήμης και θα τρέξει αυτόματα τα αποθηκευμένα δεδομένα.

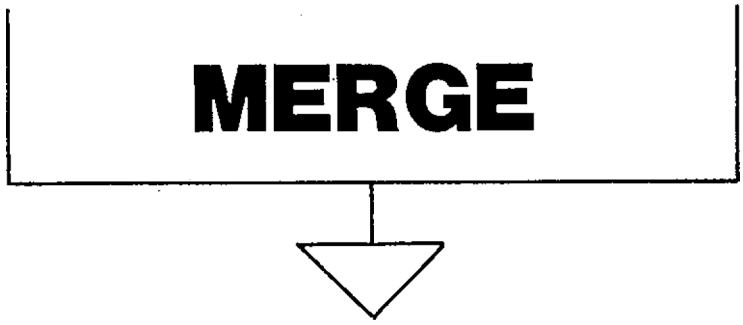
Αυτή η μορφή για σώσιμο χρησιμοποιείται σχεδόν αποκλειστικά με προγράμματα σε κώδικα μηχανής. Για να βρείτε κάτι πάνω σε γλώσσα μηχανής και assembly σας συστήνεται για διάβασμα το:

Beginners' Assembly Language Programming Course for the Amstrad.

2.1.3.4 MERGE

Τόσο καλά όσο απλά φορτώνεται ένα πρόγραμμα είναι δυνατό να συχωνεύσετε δύο προγράμματα μαζί. Ένα από τα προγράμματα πρέπει να αποθηκευτεί στη μνήμη και το άλλο στην κασέτα. Η εντολή MERGE φορτώνει το πρόγραμμα της κασέτας γραμμή με γραμμή. Υποτίθεται ότι η πρώτη γραμμή που φορτώνεται είναι η 10. Ο Amstrad ελέγχει να δει αν το πρόγραμμα στη μνήμη έχει γραμμή 10. Αν έχει, τότε αντικαθίσταται από τη γραμμή της ταινίας, αν όχι, τοτε χρησιμοποιείται η γραμμή από την ταινία. Το σχέδιο A2.1 περιγράφει τις αρχές της MERGE.

The program in the computer	The program on Tape
10 REM COMPUTER	10 REM TAPE
15 LET A=8	20 LET A=A*A
20 LET A=A+7	30 PRINT "THANK YOU";A
30 PRINT A	40 END
35 PRINT "THANK YOU"	



The Final Program

```
10 REM TAPE
15 LET A=8
20 LET A=A*A
30 PRINT "THANK YOU";A
35 PRINT "THANK YOU"
40 END
```

A Diagrammatic Representation of the Merge Command

Εικόνα A2.1

Το καινούριο πρόγραμμα είναι το πρόγραμμα της ταινίας και κάθε γραμμή του προγράμματος που ήταν στον υπολογιστή και που δεν έχει αντίστοιχο αριθμό γραμμής στο πρόγραμμα της ταινίας.

2.1.3.5 CHAIN MERGE

Η εντολή CHAIN MERGE είναι ένας ενδιαφέρον συνδυασμός των εντολών CHAIN και MERGE. Το τμήμα συγχώνευσης δουλεύει με τον τρόπο που περιγράφηκε στο σχήμα A2.1 και το τμήμα CHAIN δουλεύει ακριβώς με τον ίδιο τρόπο όπως η εντολή CHAIN μόνη της.

```
CHAIN MERGE "ALBERT"
```

Αυτό θα συγχωνεύσει το πρόγραμμα της κασέτας ALBERT με το πρόσφατο στη μνήμη πρόγραμμα. Μόλις η συγχώνευση ολοκληρωθεί το πρόγραμμα αυτόματα θα τρέξει αρχίζοντας από την πρώτη γραμμή του προγράμματος.

```
CHAIN MERGE "ALBERT",100
```

Αυτό θα συχνωνεύσει το πρόγραμμα της κασέτας ALBERT με το πρόσφατο στη μνήμη πρόγραμμα. Η εκτέλεση του προγράμματος θ' αρχίσει από τη γραμμή 100. Βεβαιώσου ότι η χρησιμοποιημένη γραμμή στην εντολή CHAIN MERGE υπάρχει. Αν όχι ο Amstrad θα αναφέρει:

```
Line does not exist
```

και το πρόγραμμα στη μνήμη θάχει διαγραφεί.

Ο συνδυασμός της CHAIN και MERGE μπορεί να χρησιμοποιηθεί και με μια τρίτη παράμετρο. Αυτή είναι η DELETE.

```
CHAIN MERGE "ALBERT",100,DELETE 300-
```

'Οταν αυτό θα εισαχθεί θ' αναγκάσει τον υπολογιστή να διαγράφει τις γραμμές από 300 και πέρα του πρόσφατου στη μνήμη προγράμματος **ΠΡΙΝ** συχνωνευτεί με το πρόγραμμα της κασέτας.

2.1.3.6 CAT

Η τελευταία από τις εντολές κασέτας του Amstrad είναι η CAT. CAT είναι η συντομογραφία της CATalogue (κατάλογος) και χρησιμοποιείται για να εμφανίσει τα ονόματα όλων των αρχείων που σώθηκαν στην ταινία του κασετοφώνου.

Μετά την πληκτρολόγηση της CAT ο Amstrad θα ανταποκριθεί με:

```
Press PLAY then any key:
```

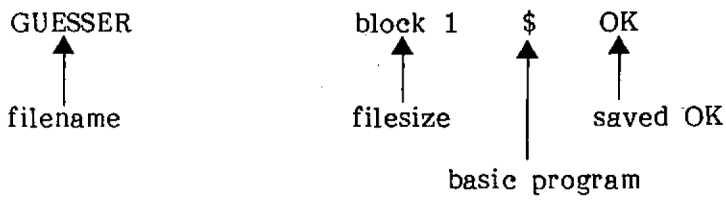
Μόλις πατηθεί ένα πλήκτρο ο Amstrad θ' αρχίσει να ψάχνει στην ταινία. Κάθε φορά που θα συναντιέται ένα πρόγραμμα θα εμφανίζονται τ' ακόλουθα:

```
<όνομα αρχείου> block <n> <τύπος αρχείου> OK
```

όπου <όνομα αρχείου> είναι το όνομα του αρχείου που βρέθηκε. <n> είναι ο αριθμός του block και <τύπος αρχείου> είναι ένας από τους παρακάτω χαρακτήρες αρχείων: sp 1

```
$ πρόγραμμα σε BASIC
% προστατευμένο αρχείο BASIC
* αρχείο με κείμενο σε ASCII
& δυαδικό αρχείο
' προστατευμένο δυαδικό αρχείο
```

Το μήνυμα OK σου λέει ότι το πρόγραμμα σώθηκε τέλεια.
Το μήνυμα καταλόγου (CAT) της ταινίας στην οποία σώθηκε
ο "GUESSER" θα μπορούσαμε να διαβάσουμε:



The CATalogue Display

Εικόνα Α2.2

3. ΛΥΣΕΙΣ

ΑΣΚΗΣΗ 1.1

* Πληκτρολόγησε EDIT 30

* Μετακίνησε τον κάρσορα μέχρι να μπει μεταξύ της PRINT και NAME$

* Πληκτρολόγησε "Your name is ";

* Πάτα ENTER

* Έκανες με επιτυχία EDIT στη γραμμή 30

ΑΣΚΗΣΗ 1.2

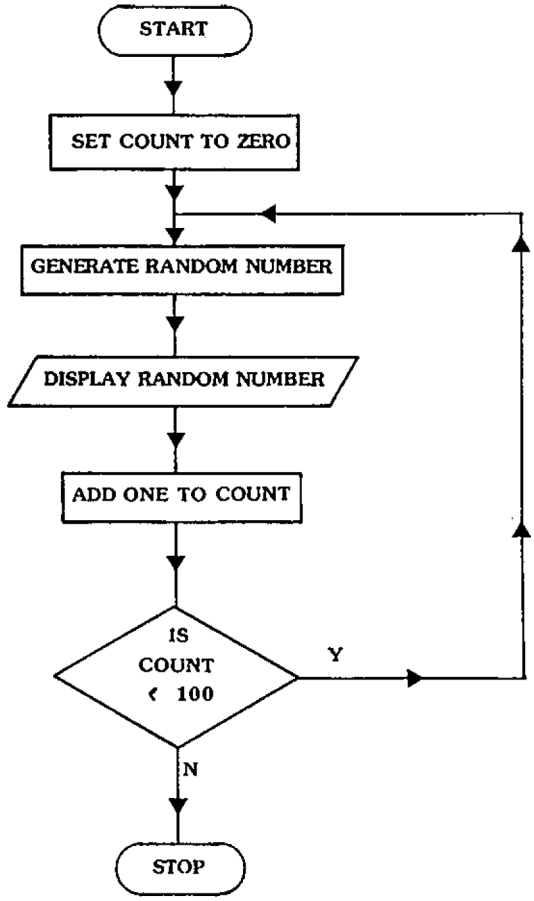
```
10 INPUT "What is your name "; NAME$
20 INPUT "How old are you "; AGE
30 PRINT "Your name is "; NAME$
40 PRINT "Your age is "; AGE
```

ΑΣΚΗΣΗ 1.3

```
10 CLS
20 INPUT "WHAT IS YOUR NAME";NAME$
30 CLS
40 LOCATE 15,10
50 PRINT "HELLO ";NAME$
```

ΑΣΚΗΣΗ 2.1

```
10 COUNT=0
30 RANUM=INT(RND*100)+1
50 PRINT RANUM
70 COUNT=COUNT+1
75 IF COUNT < 100 THEN 30
90 STOP
```



ΑΣΚΗΣΗ 2.2

```
10 FOR X=0 TO 100 STEP 3
```

ΑΣΚΗΣΗ 2.3

```
10 FOR X=10 TO 1 STEP -1
20 PRINT X
30 NEXT X
40 PRINT"FIRE!"
```

ΑΣΚΗΣΗ 2.4

One way is to change line 60.

```
60 IF GUESS=RANDOM THEN PRINT"WELL DONE-
GUESS CORRECT ":PRINT"YOU TOOK";C;"GOES":
GOTO 110
```

ΑΣΚΗΣΗ 2.5

```
100 IF COUNT <99 THEN PRINT"SORRY YOU'VE
HAD YOUR SIX GOES ":PRINT"THE NUMBER WAS";
RANUM
```

ΑΣΚΗΣΗ 3.1

```
10 MODE 0
20 BORDER 13
30 INK 0,19
40 INK 1,17
50 PAPER 0
60 PEN 1
```

ΑΣΚΗΣΗ 3.2

```
10 MODE 0:INK 1,19
20 PLOT 320,100
30 DRAW 64,93,4
```

ΑΣΚΗΣΗ 3.3

```
75 IF A$="R" then 30
```

ΑΣΚΗΣΗ 3.4

```
3 IF X>640 OR X<0 THEN PRINT"NUMBER OUT
OF RANGE":GOTO 2

5 IF Y>400 OR Y<0 THEN PRINT"NUMBER OUT
OF RANGE":GOTO 4
```

ΑΣΚΗΣΗ 3.5

```
10 MODE 1
20 R=57
25 PLOT 177,200
30 FOR A=0 TO 3.15
40 X= R*COS(A):Y=R*SIN(A)
50 DRAW 120+X,200+Y
60 NEXT A
```

ΑΣΚΗΣΗ 4.1

```
10 FOR X=1 TO 4
20 READ A,A$
30 PRINT A;A$
40 NEXT X
50 DATA 1,ONE,2,TWO,3,THREE,4,FOUR
```

3.1 ΠΑΡΑΡΤΗΜΑ 1

ΑΣΚΗΣΗ Α1.1

i) 0000 0011 = 0+0+0+0+0+0+2+1
= 3

ii) 0000 0100 = 0+0+0+0+0+4+0+0
= 4

iii) 1000 0000 = 128+0+0+0+0+0+0+0
= 128

iv) 1000 0011 = 128+0+0+0+0+0+2+1
= 131

v) 1011 0111 = 128+0+32+16+0+4+2+1
= 183

vi) 0111 0011 = 0+64+32+16+0+0+2+1
= 115

ΑΣΚΗΣΗ Α1.2

i) 0009_{16} = 0x409+0x256+0x16+9x1
= 0+0+0+9
= 9_{10}

ii) 0013_{16} = 0x4096+0x256+1x16+3x1
= 0+0+16+3
= 19_{10}

iii) $00A5_{16}$ = 0+0+10x16+5x1
= 160+5
= 165_{10}

iv) $0AAE_{16}$ = 0+10x256+10x16+14x1
= 2560+160+14
= 2734_{10}

v) $000E_{16}$ = 0+0+0+14
= 14_{10}

vi) $011A_{16}$ = 0+256+16+10
= 282_{10}

vii) $00EA_{16}$ = 0+0+14x16+10
= 224+10
= 234_{10}

ΑΣΚΗΣΗ Α1.3

i) 4_{10} = 0100_2 (BCD)

ii) 10_{10} = 1x10+0

iii) 77_{10} = 7x10+7
= $0111\ 0111_2$ (BCD)

iv) 97_{10} = 9x10+7
= $1001\ 0111_2$ (BCD)

v) 53_{10} = 5x10+3
= $0101\ 0011_2$ (BCD)

vi) 102_{10} = 1x100+0x10+2x1
= $0001\ 0000\ 0010_2$ (BCD)

vii) 953_{10} = 9x100+5x10+3x1
= $1001\ 0101\ 0011_2$ (BCD)

viii) 2579 = 2x1000+5x100+7x10+9x1
= $0010\ 0101\ 0111\ 1001_2$ (BCD)

ΑΣΚΗΣΗ Α1.4

i) $0000\ 0001_2$ (BCD)= 0x10+1x1
= 1_{10}

ii) $0000\ 1001_2$ (BCD)= 0x10+9x1
= 9_{10}

iii) $0001\ 0101_2$ (BCD)= 1x10+5x1
= 15_{10}

iv) $0010\ 0000_2$ = 2x10+0x1
= 20_{10}

v) $0100\ 1001_2$ (BCD)= 4x10+9x1
= 49_{10}

vi) $1010\ 0011_2$ (BCD)

*** This is not a valid BCD number as the first nybble, 1010_2 = 10_{10} , i.e. is greater than ,allowed in BCD.

vii) $1001\ 0111_2$ (BCD)= 0x10+7x1
= 97_{10}

viii) $1000\ 1000_2$ (BCD)= 8x10+8x1
= 88_{10}

UNIVERSITY STUDIO PRESS
Εκδόσεις Επιστημονικών Βιβλίων & Περιοδικών
Κων. Μελενίκου 13/15 τηλ. 209637 & 209837 Θεσσαλονίκη

AMSTRAD 1

CPC 464, 6128

ΑΡΧΙΖΟΝΤΑΣ BASIC

Το βιβλίο αυτό είναι γραμμένο για κάθε αρχάριο χρήστη υπολογιστή που θέλει να μάθει τη γλώσσα BASIC.

Το βιβλίο αυτό οδηγεί τον αναγνώστη βήμα προς βήμα μέσα στη BASIC του Amstrad. Οι εντολές αναπτύσσονται αναλυτικά καθώς χρησιμοποιούνται σε πολλά παραδείγματα προγραμμάτων. Ο αναγνώστης θα έχει όλες τις απαραίτητες γνώσεις στο τέλος του Α΄ κεφαλαίου ώστε να μπορεί να γράφει τα δικά του απλά προγράμματα.

Τα προγράμματα του βιβλίου έχουν γραφτεί με τη χρήση του δομημένου προγραμματισμού, για πιο εύκολη κατανόηση της δημιουργίας προγραμμάτων.

Το βιβλίο αυτό είναι ένα πλήρες μάθημα αυτοδιδασκαλίας προγραμματισμού BASIC αλλά αν θέλετε να μάθετε περισσότερα, το δεύτερο βιβλίο με τίτλο «Amstrad 2 - ΗΧΟΣ, ΓΡΑΦΙΚΑ, ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ», θα σας βοηθήσει.

GLENTOP publishers

MPS

ΠΛΗΡΟΦΟΡΙΚΗ Β. ΕΛΛΑΔΟΣ
ΠΟΛΥΤΕΧΝΕΙΟΥ 47, ΤΗΛ. 540 246
546 25 - ΘΕΣΣΑΛΟΝΙΚΗ

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/