Τι είναι το debug μαμα?
Με απλά λόγια, debug (αποσφαλμάτωση αλλά που να κάθεσαι να το πεις), είναι η διαδικασία εντοπισμού και διόρθωσης σφαλμάτων που υπάρχουν σε κώδικα (ασχέτως γλώσσας προγραμματισμού). Δεδομένου ότι ο εντοπισμός των λαθών είναι αρκετά χρονοβόρος (και αρκετά κουραστικός τις περισσότερες φορές) έχουν δημιουργηθεί κατάλληλα εργαλεία προκείμενου να βοηθούν τη διαδικασία, αλλά όχι να την κάνουν αυτόματα! Το debugging, δυστυχώς, δεν είναι click-on "debug this stuff" και τα κάνει όλα μόνο του.. Μακάρι, αλλά που τέτοια τύχη! :(

Τι μπορεί και τι ΔΕΝ μπορεί να κάνει το debug?
Το debug ενός προγράμματος μπορεί να διορθώσει σφάλματα τα οποία συνήθως παρουσιάζονται κατά το runtime του προγράμματος. Δηλαδή, το debug βοηθάει από τη στιγμή που το πρόγραμμα έχει περάσει το στάδιο του compilation, το έχουμε βάλει να τρέξει και εκεί που τρέχει πετάει μήνυμα λάθους. Επίσης, βοηθάει στο να βρούμε γιατί δεν παίρνουμε τα σωστά αποτελέσματα, η τουλάχιστον αποτελέσματα τα οποία θα περιμέναμε να βρούμε. Αυτό που ΔΕΝ κάνει το debug είναι να σε βοηθήσει να φτιάξεις κάποιον αλγόριθμο για την επίλυση του αρχικού σου προβλήματος. Θα σε βοηθήσει π.χ. να *καταλάβεις* τι θα έπρεπε να είχες γράψει σε μια γραμμή κώδικα , ή το γιατί δεν δουλεύει ένα συγκεκριμένο κομμάτι κώδικα όπως θα ήθελες, αλλά δε θα σου πει ΤΙ έπρεπε να είχες γράψει για να πάρεις το επιθυμητό αποτέλεσμα.

Μπορώ να κάνω debug?
Ναι, αρκεί να έχεις και μια πλατφόρμα ανάπτυξης κώδικα. Το ότι υπάρχουν εργαλεία για να σε βοηθήσουν να κάνεις debug δε σημαίνει ότι χωρίς αυτά τα εργαλεία δεν γίνεται debug.

Πώς γίνεται το debug?
Το debug γίνεται με δύο τρόπους, ως επί το πλείστον. Τουλάχιστον, εγώ δύο έμαθα, και αυτούς αναφέρω.

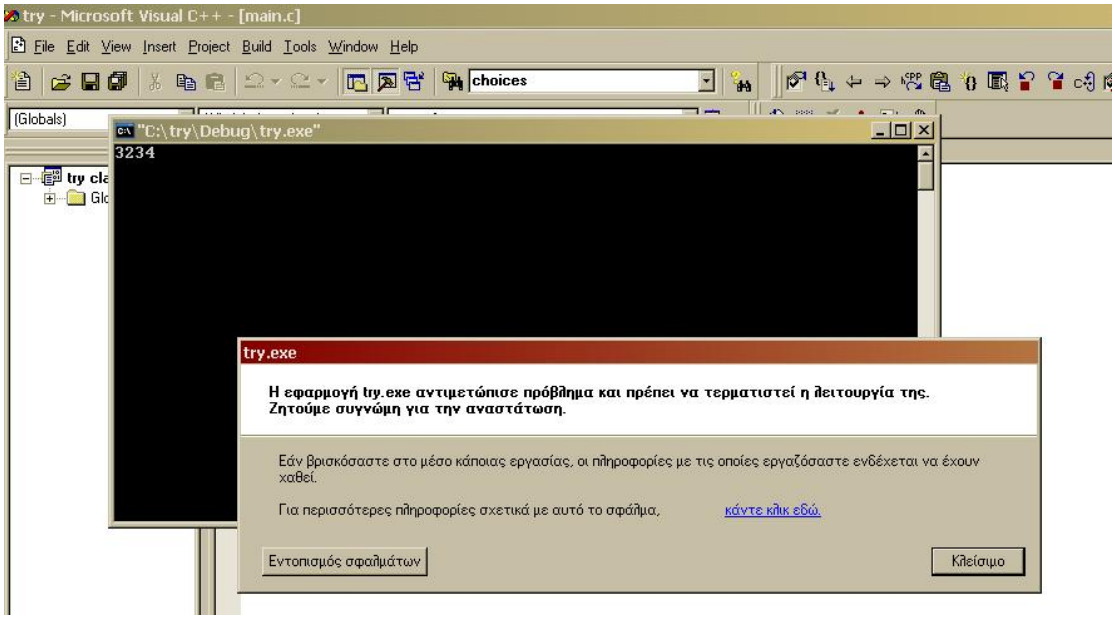
1) Ο αρχικά δύσκολος τρόπος, που παραμένει δύσκολος
Κλασσική μέθοδος για το debugging κώδικα είναι να τυπώνεις στην οθόνη κάποια μηνύματα λάθους, του στυλ "Λάθος 1", "λάθος 2" κλπ, ή ό,τι άλλο μήνυμα αρέσει στον καθένα. Ο τρόπος αυτός είναι δύσκολος για το λόγο ότι η εντολή εκτύπωσης (πχ printf, cout, System.out.println()) μπαίνει κάτω από κάθε εντολή που έχει γράψει κάποιος. Προφανώς δε χρειάζεται να μπαίνει κάτω από τετριμμένες εντολές, όπως i = i + 1 ή μια άλλη εντολή εκτύπωσης στην οθόνη. Για το λόγο αυτό, η συγκεκριμένη μέθοδος δεν δίνει και τα καλύτερα αποτελέσματα, γιατί εντοπίζεται το λάθος "στο περίπου". Παρ' όλα αυτά, το καλό της μεθόδου αυτής είναι ότι μπορεί να χρησιμοποιηθεί ανεξάρτητα πλατφόρμας (είτε κάποιος

δουλεύει σε VS, είτε σε DEV είτε σε Unix περιβάλλον ή δεν ξέρω κι εγώ σε τι άλλο :)). Έτσι, μπορείς να πεις π.χ. ότι αφού μου τύπωσε στην οθόνη την τάδε γραμμή αλλά όχι την άλλη, το λάθος έγινε μεταξύ αυτών των δύο, η τουλάχιστον κάπου εκεί κοντά. Αυτό δυστυχώς δεν ισχύει πάντα, για το λόγο αυτό ακολουθεί άλλη διαδικασία, μετά τον εντοπισμό του "περίπου λάθους". Αυτή περιλαμβάνει το να τυπωθούν στο σημείο του "περίπου λάθους" οι διάφορες μεταβλητές που χρησιμοποιούνται από το πρόγραμμα, ώστε να πάρουμε τις τιμές. Προφανώς κάτι τέτοιο δεν είναι και το ευκολότερο πράγμα στον κόσμο.... φανταστείτε έναν πίνακα 100 θέσεων και αρκετές ακόμα μεταβλητές, να τυπώνονται στην οθόνη... Καλό λοιπόν είναι όταν γίνεται το debugging με αυτή τη μέθοδο, να υπάρχει μια διαίσθηση του "τι ψάχνω? (ω εω! :p)" προκειμένου να τυπώνονται μόνο οι τιμές των μεταβλητών που πιστεύουμε ότι είναι λανθασμένες, διαφορετικά δε θα βοηθήσει σε τίποτα η μέθοδος αυτή.

2η μέθοδος, που περιλαμβάνει τη χρήση του debugger (εργαλείου που παρέχεται συνήθως μαζί με την πλατφόρμα ανάπτυξης)
Από εδώ και πέρα αναφέρομαι μόνο σε VS (ελπίζω οι εντολές να είναι με παρόμοιο τρόπο και στις άλλες windows based πλατφόρμες). Ο λόγος που χρησιμοποιήθηκε το VS είναι ότι αυτό ξέρω να χρησιμοποιώ (όχι δεν πληρώνομαι γι' αυτό που κάνω)… Για να γίνουν κατανοητά τα όσα λέω, θα χρησιμοποιήσω κάποια screenshots προκειμένου να υπάρχει και οπτικοποίηση του σε τι αναφέρομαι. Θα προσπαθήσω να φέρω μερικά απλά παραδείγματα, για να δείξω το πως δουλεύει ο debugger του VS.

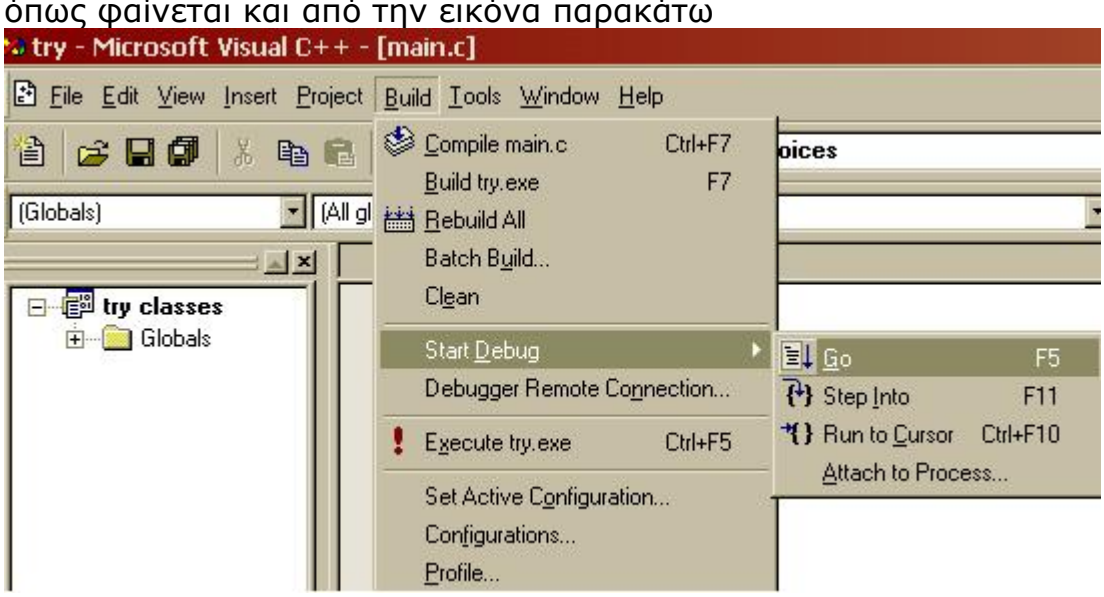
Για να δουλέψει ο debugger χρειάζεται να υπάρχει κάποιο project.

* Πώς ξεκινάει ο debugger?
Έστω ότι τρέχοντας την εφαρμογή (είπαμε, έχουμε περάσει το στάδιο του compilation) βγαίνει το εξής ωραίο μήνυμα:



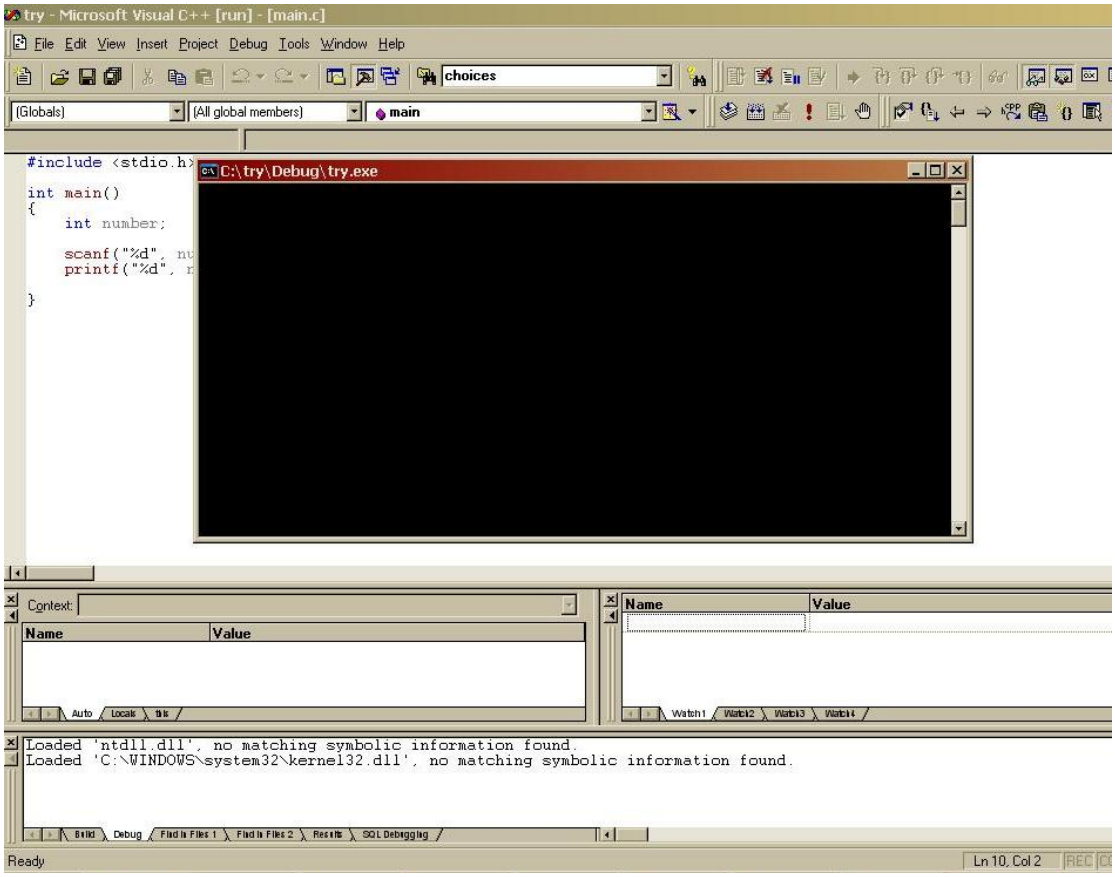
Βλέποντας αυτό το μήνυμα λοιπόν, ανοίγουμε τον debugger, ως εξής:

Build -> Debug -> Go
όπως φαίνεται και από την εικόνα παρακάτω



Το F5 που έχει δίπλα, είναι η συντόμευση για να αρχίσει το debug mode.

Επιλέγοντας λοιπόν το Go, το VS περνάει σε debug mode, that looks like this:



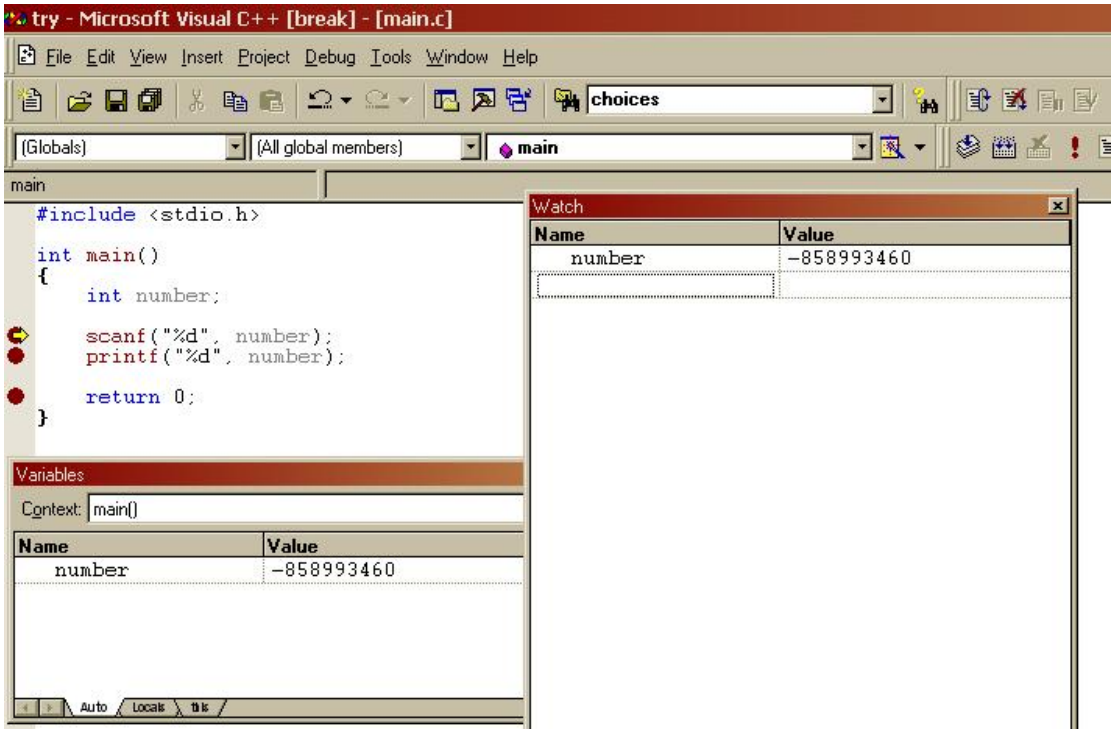
Συγκεκριμένα, ανοίγει η κονσόλα που θα άνοιγε σε περίπτωση που τρέχαμε κανονικά το πρόγραμμα. Στο μεγάλο παράθυρο πίσω από την κονσόλα είναι ο κώδικας που έχουμε γράψει για το πρόγραμμα. Από τα 3 κάτω παράθυρα, αυτό που ενδιαφέρει περισσότερο είναι το πάνω δεξιά, που έχει 4 καρτέλες με ονόματα watch1, watch2, watch3, watch4. Το συγκεκριμένο παράθυρο, είναι το "**watch**" (ω! τι παράξενο ☺) και ο λόγος ύπαρξής του είναι να παρακολουθεί τις τιμές των μεταβλητών. Τα άλλα 2 παράθυρα περιέχουν πληροφορίες που δεν εξυπηρετούν ιδιαίτερα το χρήστη, επομένως και δεν αφορούν άμεσα. Αν δε σας βολεύει μπορείτε να αλλάξετε τη θέση των παραθύρων προκειμένου να εξυπηρετήστε εσείς οι ίδιοι (όπως θα κάνω και εγώ σε λίγο).

Παρατήρηση: Τα χρώματα που εμφανίζονται να έχουν οι συναρτήσεις ΔΕΝ είναι του debugger. Αν σε εσάς δεν υπάρχουν, μην παραξενευτείτε γιατί.

<u>Πως χρησιμοποιείται ο debugger</u>

Υπάρχουν τα λεγόμενα breakpoints, το οποία ουσιαστικά λένε στον debugger πού πρέπει να σταματήσει. Βασικά, δε θα σταματήσει να δουλεύει, απλά θα εκτελέσει τον κώδικα μέχρι εκείνο το σημείο (όπου δηλαδή υπάρχει το breakpoint) και θα περιμένει κάποια

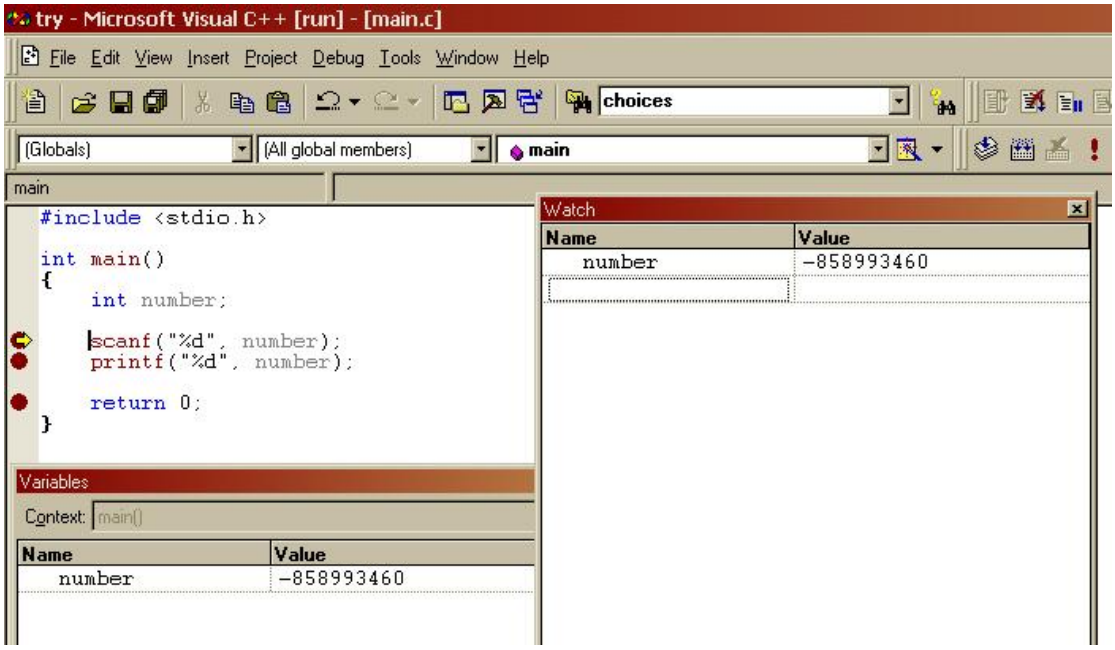
απόκριση από τον χρήστη. Αν βάλετε κάποιο breakpoint σε ένα σημείο, στη αντίστοιχη γραμμή μπροστά θα εμφανιστεί μια κόκκινη κουκίδα.



Η καρτέλα **Variables** είναι σε auto mode, δηλαδή θα αλλάζει το περιεχόμενο της αυτόματα, δείχνοντας τις τελευταίες αλλαγές που έγιναν. Όπως φαίνεται και από την παραπάνω εικόνα, στην καρτέλα **watch** έχω συμπεριλάβει τη μεταβλητή number. Στα αριστερά φαίνεται το όνομα της μεταβλητής ενώ στα δεξιά φαίνεται η τιμή που έχει η μεταβλητή τη χρονική στιγμή εκείνη. Προκειμένου να παρακολουθήσετε την τιμή μιας μεταβλητής, αρκεί απλά να γράψετε το όνομα της μεταβλητής στα αριστερά κουτιά (κάτω από εκεί που γράφει **name**). Η τιμή που φαίνεται σε αυτή τη φάση (ο πολύ μεγάλος αρνητικός αριθμός) είναι η default τιμή που παίρνει μια μεταβλητή τύπου int κατά τη δήλωσή της. Όταν η τιμή μιας μεταβλητής αλλάζει, οι αλλαγές γίνονται <u>αυτόματα</u> στο παράθυρο του watch. Δε χρειάζεται να κάνετε refresh η κάτι σχετικό. Επίσης, όταν αλλάζει η τιμή μιας μεταβλητής, αυτό φαίνεται με κόκκινο. Δηλαδή,

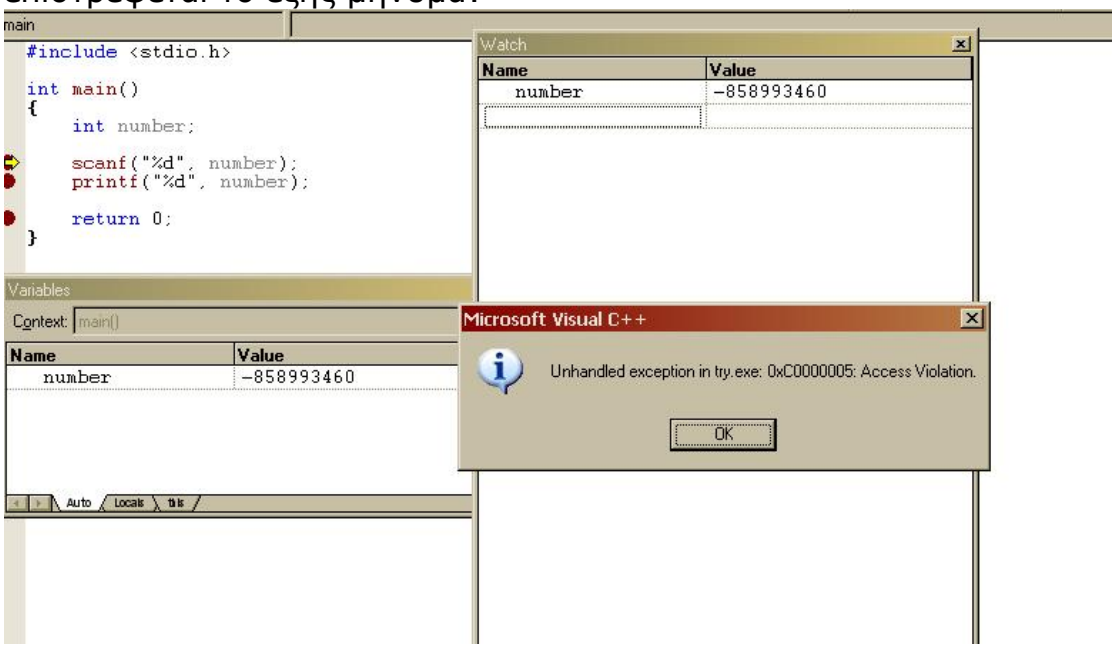
Προκειμένου να τρέξετε μέχρι το breakpoint τον κώδικά σας, πηγαίνετε και επιλέγετε:

Debug -> Go (ή πατήστε απλά το F5) (δε θα βάλλω και για αυτό εικόνα, μην τρελαθούμε!)



Παρατηρήστε ότι στο παράθυρο **Variables**, εκεί που λέει Context (main()), η περιοχή έχει γκριζαριστεί, ενώ πριν ήταν άσπρη. Αυτό σημαίνει ότι το πρόγραμμα περιμένει κάποια ενέργεια από τον χρήστη (το πιθανότερο κάποια είσοδο από την κονσόλα). Ακόμα και αν δοκιμάσετε να πάτε στην επόμενη εντολή (Debug -> step over), αυτό δε θα γίνει, γιατί το πρόγραμμα περιμένει κάποια ανταπόκριση.

Δίνουμε λοιπόν ένα νούμερο στην κονσόλα (αφού αυτό περιμένει το πρόγραμμα σε αυτή τη φάση), και πατώντας το enter μας επιστρέφεται το εξής μήνυμα:



Παίρνοντας αυτό το μήνυμα αντιλαμβανόμαστε, κοιτάζοντας τον κώδικά μας, σε ποιο σημείο έχει «σκάσει» (μπαμ! :p) το πρόγραμμα. Το σημείο στο οποίο έχει μείνει το κίτρινο βελάκι δείχνει ακριβώς την

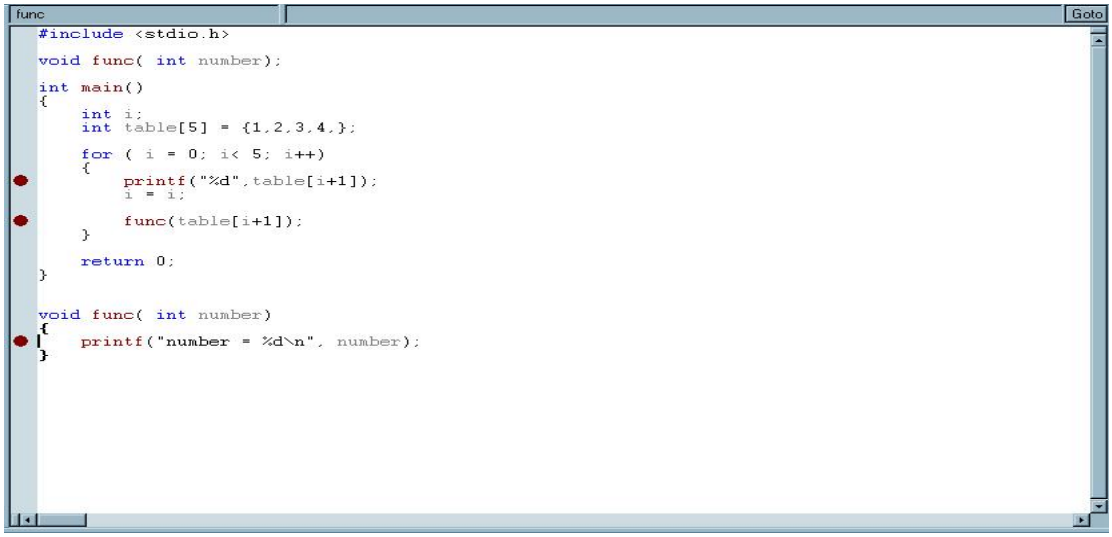
γραμμή εκείνη στην οποία υπάρχει το πρόβλημα. Για το λόγο αυτό, προκειμένου να διευκολυνόμαστε και στο debugging (πέραν της αναγνωσιμότητας του προγράμματος), γράφουμε μία – μία τις εντολές ανά γραμμή. Ο λόγος είναι ότι το επόμενο breakpoint είναι σε επόμενη γραμμή, επομένως κάνοντας "Debug->Go" θα εκτελεστούν όλες οι εντολές της γραμμής, και άρα το λάθος μπορεί να «ξεφύγει».

Παρατηρώντας την παραπάνω εικόνα, και το σημείο στο οποίο έχει «κολλήσει» το κίτρινο βελάκι (στην scanf()) παρατηρούμε ότι λείπει το σύμβολο & από τη μεταβλητή number. Κάνουμε την αλλαγή και ξανατρέχουμε το πρόγραμμα (το οποίο θα πρέπει να τρέξει κανονικά).

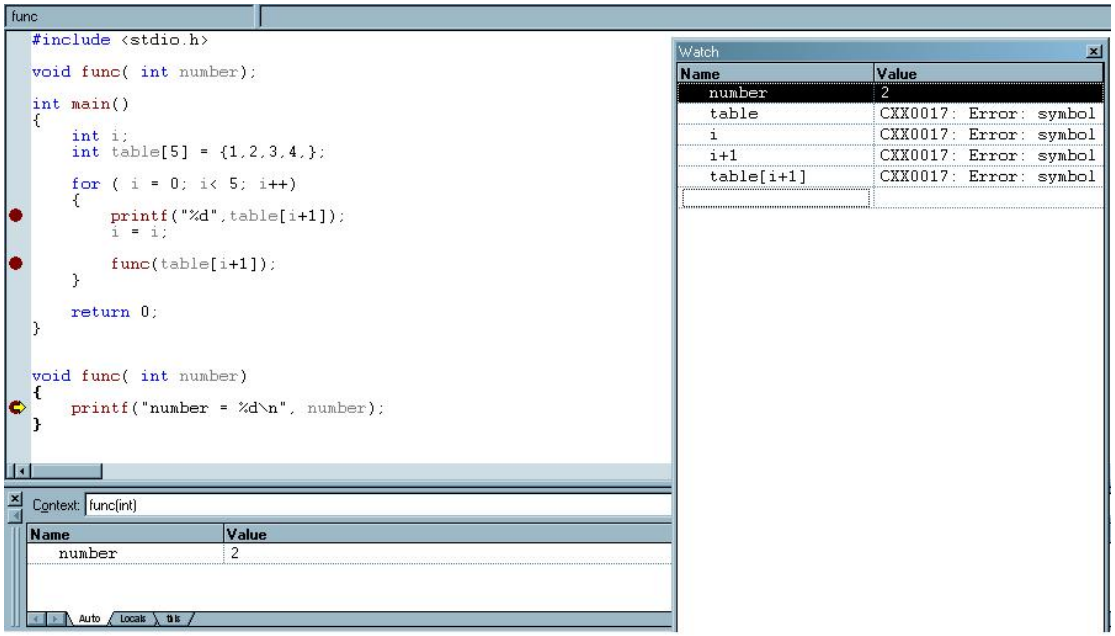
Καλά τα όσα είπαμε μέχρι εδώ, αλλά τι γίνεται στην περίπτωση που το πρόγραμμά μου έχει σπάσει σε συναρτήσεις, για να το δουλέψω καλύτερα? Κανένα πρόβλημα! Η λύση είναι ότι θα μπω να εξετάσω τη συνάρτηση εσωτερικά.

Μα καλά! Πώς θα το κάνω αυτό?

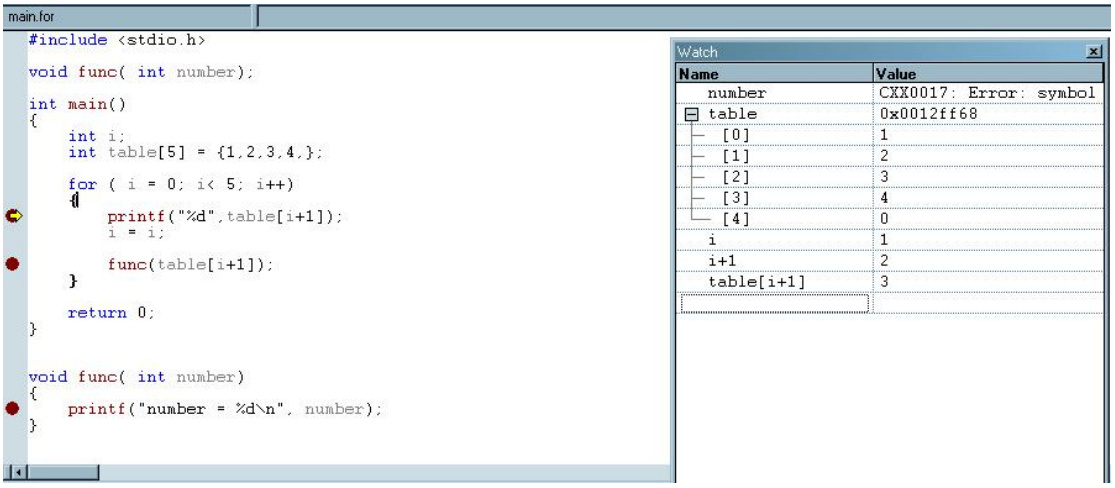
Μα χρησιμοποιώντας και πάλι τα break points! Τοποθετούμε ένα breakpoint μέσα στη συνάρτηση, στο σημείο στο οποίο επιθυμούμε να σταματήσει το πρόγραμμά μας. Ένα τέτοιο παράδειγμα φαίνεται στην επόμενη εικόνα.



Αν κάποιος δοκιμάσει να τρέξει το παραπάνω πρόγραμμα, θα διαπιστώσει ότι δεν παίρνει σωστά αποτελέσματα (το σωστά εδώ αναφέρεται στο να τυπώσει τις τιμές του πίνακα table). Κάνοντας το debug θα διαπιστώσει ότι οι τιμές που περνάνε στην συνάρτηση func, είναι λανθασμένες. Προφανώς, αφού στη συνάρτηση μέσα δε γίνεται τίποτα περισσότερο από το να τυπώνεται η μεταβλητή, καταλαβαίνει κανείς ότι έχει περαστεί λάθος όρισμα στη συνάρτηση. Επομένως, το πρόβλημα είναι κατά την κλήση της συνάρτησης (δηλαδή στο func(table[i+1]);). Όταν η εκτέλεση του κώδικα έχει φτάσει μέσα στη συνάρτηση, ο debugger μοιάζει κάπως έτσι:



Παρατηρήστε ότι στο παράθυρο **Watch** η μεταβλητή number έχει δίπλα την τιμή 2. Η τιμή αυτή είναι η τιμή που περάστηκε σε αυτή τη φάση του τρεξίματος ως παράμετρος στη συνάρτηση func. Αντιθέτως, οι μεταβλητές table, i, i+1, table[i+1] δεν έχουν κάποια τιμή γιατί πολύ απλά δεν έχουν οριστεί μέσα στη συνάρτηση func. Αν οριστεί μια μεταβλητή I μέσα στην func, τότε στο παράθυρο **Watch**, η αντίστοιχη μεταβλητή I θα «δείχνει» το περιεχόμενο της μεταβλητής που ορίστηκε στην func και όχι αυτή που υπάρχει στην main. Αντίστοιχα, όταν θα επιστραφεί ο έλεγχος στην main (δηλαδή, αν από το σημείο στο οποίο είμαστε πατήσουμε Debug->Go ή F5), η μεταβλητή number δε θα έχει κάποια τιμή, ενώ θα αποτιμηθούν οι υπόλοιπες μεταβλητές.



Προφανώς, το παραπάνω λάθος έγκειται στο ότι δεν περνάει ως όρισμα το table[i] αλλά το table[i+1]. Αν αλλάξουμε το όρισμα το οποίο περνάμε, θα παίρνουμε τα σωστά αποτελέσματα.

Αυτά λίγο πολύ για το πώς δουλεύει ο debugger.

Τα συγκεκριμένα είναι δυο πολύ απλά παραδείγματα για να δει κάποιος το πώς δουλεύει. Χρησιμοποιώντας τον θα βρείτε και εσείς οι ίδιοι το πώς θα σας χρησιμεύσει.

Πέραν των breakpoints, οφείλω να αναφέρω τα shortcuts τα οποία μπορεί κάποιος να βρει ιδιαίτερα χρήσιμα. **Άσχετα** από το αν έχετε βάλει breakpoints ή όχι, μπορείτε να χρησιμοποιήσετε τα shortcuts που αναφέρω, για να γλιτώνετε και χρόνο και όρεξη!

Ctrl+F10 : Το πρόγραμμα κάνει run_to_cursor (τρέχει δηλαδή μέχρι εκεί που είναι ο κέρσορας), ανεξαρτήτως αν υπάρχουν while, if, for, συναρτήσεις που πρέπει να εκτελεστούν και άλλα. Συγκεκριμένα, μπορείτε να τοποθετήσετε τον κέρσορα μέσα σε μία συνάρτηση και με το shortcut αυτό, θα φροντίσει ο υπολογιστής να τρέξει το πρόγραμμα ολόκληρο, να μπει στη συνάρτηση στην οποία είναι ο κέρσορας και να φτάσει στο σημείο εκείνο. Ομοίως, μπορείτε να τοποθετήσετε τον κέρσορα μέσα σε μια if (η οποία μπορεί να βρίσκετε σε κάποιο while, ή συνάρτηση ή οπουδήποτε). Θα σας εμφανιστεί ιδιαίτερα χρήσιμο όταν ξέρετε ότι όλα τα υπόλοιπα μέχρι εκείνο το σημείο τρέχουν σωστά.

F10 : Εκτελεί την εντολή (ή τις εντολές) που βρίσκονται στη γραμμή όπου είναι το κίτρινο βελάκι. Το βελάκι τοποθετείται στην επόμενη γραμμή. Η «εντολή» μπορεί να είναι και κλήση συνάρτησης. Στην περίπτωση αυτή, δε θα μπει στη συνάρτηση μέσα, παρά θα προσπεράσει τη συνάρτηση σαν να ήταν μία εντολή. Με το F10

ελέγχει κάποιος βήμα προς βήμα τη ροή του προγράμματος (μπορεί κάλλιστα να χρησιμοποιηθεί και σε if, και σε while και οπουδήποτε).

F11 : Είναι η λύση στο πρόβλημα του F10 – όταν συναντάται μία συνάρτηση, μπορεί να πατηθεί το F11 προκειμένου να μεταπηδήσει ο έλεγχος στο εσωτερικό της συνάρτησης, αντί η συνάρτηση να εκτελεστεί ως «απλή» εντολή. Μέσα στη συνάρτηση φυσικά χρησιμοποιείται το F10 πλέον για εκτέλεση απλών εντολών, εκτός και αν υπάρχει και άλλη συνάρτηση μέσα στη συνάρτηση, οπό και χρησιμοποιείται εκ νέου το F11 κ.ο.κ.

Triggered by lebi :p :D
Ευχαριστώ τα άτομα που με έμαθαν να κάνω debugging στα προγράμματά μου!. ☺