Εργαστήριο 0ο

Εγκατάσταση

Πρώτη χρήση του Visual Studio

Χρήση του debugger

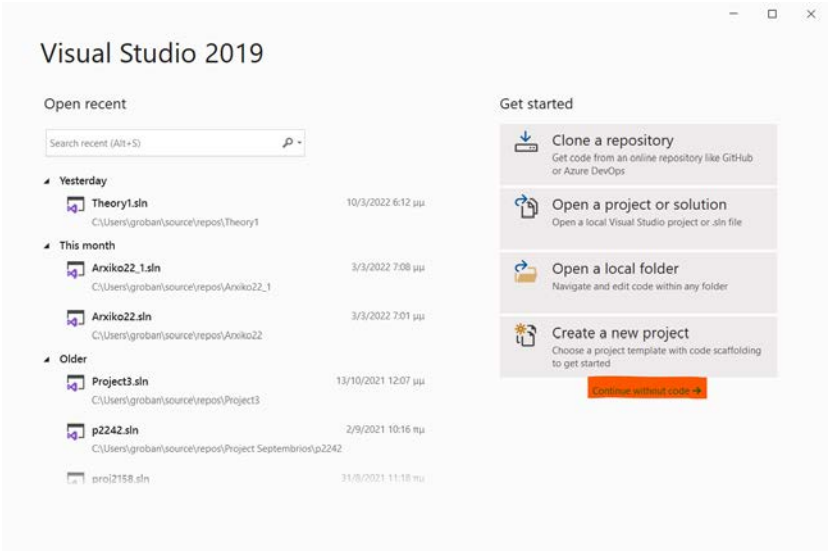
1. Εγκατάσταση του Visual Studio Community

Για την δημιουργία προγραμμάτων C++ κατά την διάρκεια του μαθήματος θα χρησιμοποιηθεί το **Microsoft Visual Studio (MSVC)** και μάλιστα η έκδοση **Community** η οποία διατίθεται δωρεάν από την εταιρία Microsoft.

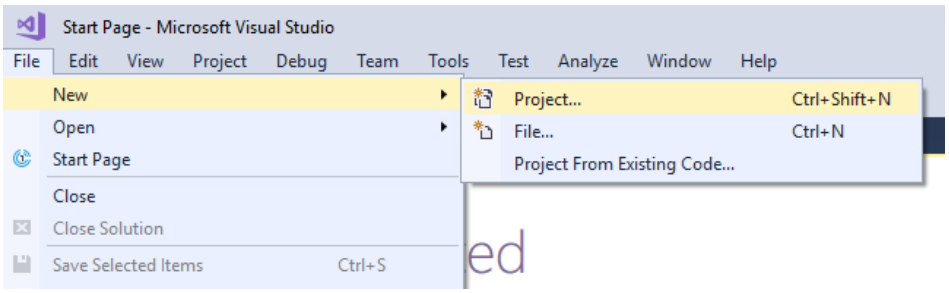
Κατά την διάρκεια της εγκατάστασης και όταν εμφανίσει το παράθυρο επιλογής **Installing – Workloads** επιλέγουμε το "**Desktop development with C++**." (μπορούμε στο μέλλον να προσθέσουμε / αφαιρέσουμε και άλλα τμήματα του MSVC όπως για παράδειγμα η δημιουργία προγραμμάτων που θα εκτελούνται σε λειτουργικό σύστημα Linux)

2. Δημιουργία ενός προγράμματος C++ στο MSVC

Με την εκτέλεση του MSVC εμφανίζεται ένα παράθυρο όπου μπορούμε να εκτελέσουμε τις βασικές λειτουργίες όπως η δημιουργία ενός καινούργιου project ή το άνοιγμα ενός υπάρχοντος project. Αν θέλουμε να προχωρήσουμε στο πραγματικό περιβάλλον του MSVC επιλέγουμε το link "Continue without code"



Για την δημιουργία ενός οποιοσδήποτε προγράμματος πρέπει να δημιουργήσουμε ένα project. (**File->New->Project**). Ένα project είναι η ονομασία που δίνει το MSVC για να ομαδοποιήσει όλα τα αρχεία (πηγαίος κώδικας, βιβλιοθήκες, εικόνες κ.λ.π.) καθώς και τις ρυθμίσεις που χρειάζονται για την μεταγλώτιση τους με σκοπό την δημιουργία ενός εκτελέσιμου προγράμματος.

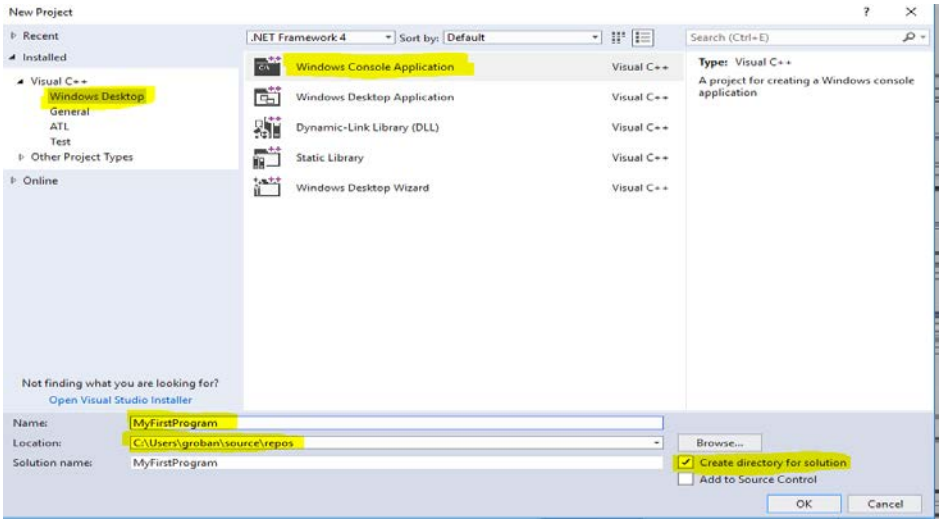


Στο επόμενο παράθυρο εκτελούμε τις εξής επιλογές (είναι τα πεδία που είναι highlighted)

- Στο αριστερό τμήμα **Visual C++ -> Windows Desktop**
- Στο δεξί τμήμα **Windows Console Application**
- Κάτω στο πεδίο Name το όνομα της εφαρμογής (π.χ. MyFirstProgram)
- Και στο πεδίο **Location** τον φάκελο στον οποίο το MSVC θα αποθηκεύσει τον πηγαίο κώδικα της εφαρμογής μας.
- Επίσης να είναι επιλεγμένο και το πεδίο "**Create directory for the solution**" για να δημιουργήσει έναν κατάλογο που θα περιέχει όλα τα αρχεία του προγράμματος (γιατί όπως αναφέρθηκε προηγουμένως είναι περισσότερα από ένα)

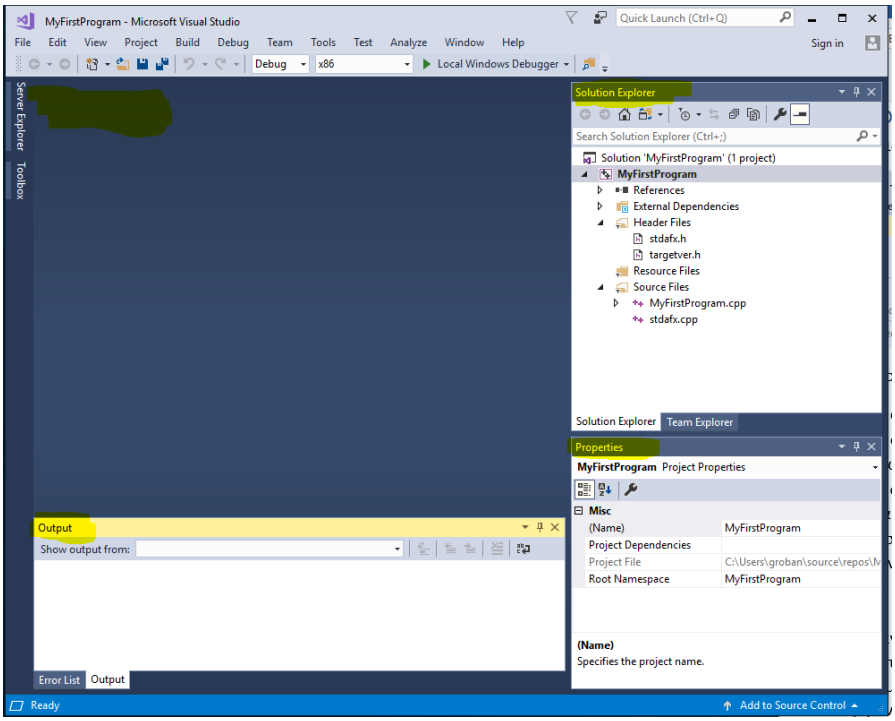
Το Console Application δημιουργεί αυτόματα ένα αρχείο Source.cpp και εισάγει τον κώδικα για την συνάρτηση main

Αν επιλέξετε Empty Project θα πρέπει να τα δημιουργήσετε μόνοι σας



Δημιουργία του κώδικα

Στο η επιφάνεια εργασίας του MSVC αποτελείται από 4 μέρη. Το πρώτο μέρος, που προς το παρόν είναι κενό, είναι αυτό που εμφανίζει τον κώδικα της εφαρμογής, το δεύτερο (Solution Explorer) όπου περιέχει τα αρχεία που project, το τρίτο που εμφανίζει ιδιότητες του στοιχείου που έχετε επιλεγμένο κάθε φορά και τέλος το Output όπου εμφανίζονται πληροφορίες σχετικές με την μεταγλώτιση (compiling) του προγράμματος.

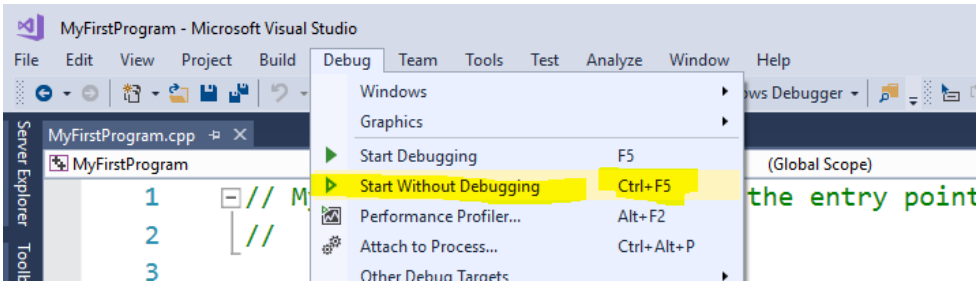


Για να εμφανιστεί ο κώδικας της main συνάρτησης κάνουμε διπλό κλίκ στο αρχείο MyFirstProgramm.cpp που εμφανίζεται το παράθυρο του Solution Explorer.

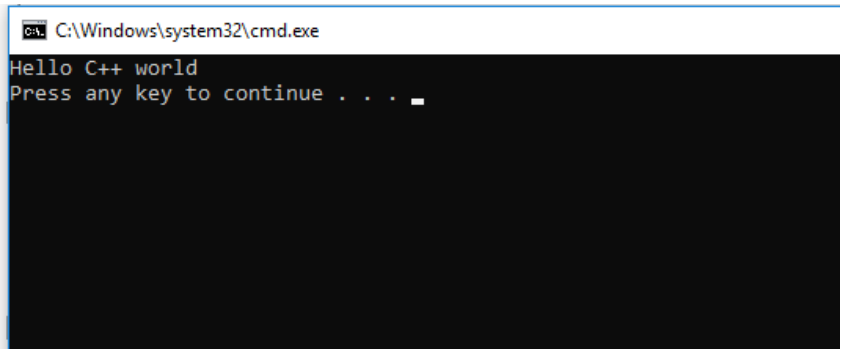
Στο αρχείο που εμφανίζεται προσθέτετε τον παρακάτω κώδικα (τις γραμμές που είναι τονισμένες με πολύ προσοχή)

```
#include <iostream>
using namespace std;


int main()
{
    cout<<"Hello C++ world\n";
    return 0;
}
```

Για την εκτέλεση του προγράμματος επιλέγουμε **Debug->Start without debugging** (ή τα πλήκτρα **Ctrl + F5**).

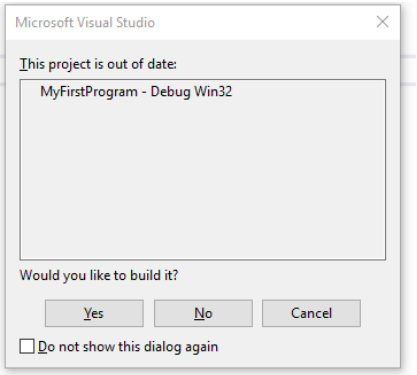


Αν το πρόγραμμα μας δεν περιέχει συντακτικά λάθη τότε θα εμφανιστεί το τερματικό όπου θα εκτελείται.



Διόρθωση συντακτικών λαθών

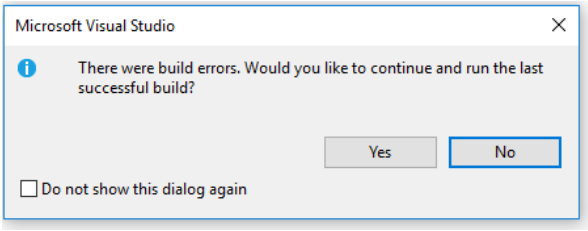
Κάθε φορά που κάνουμε κάποια διόρθωση στον κώδικα και προσπαθούμε να το εκτελέσουμε (είτε με debug->Start είτε με τα πλήκτρα) εμφανίζεται το παρακάτω μήνυμα



Αυτό σημαίνει ότι έχουν πραγματοποιηθεί αλλαγές στον κώδικα και ότι χρειάζεται το MSVC να τον ξανακάνει compile προκειμένου να τον εκτελέσει. Επιλέγουμε **πάντα το Yes**

ΠΡΟΣΟΧΗ!!!

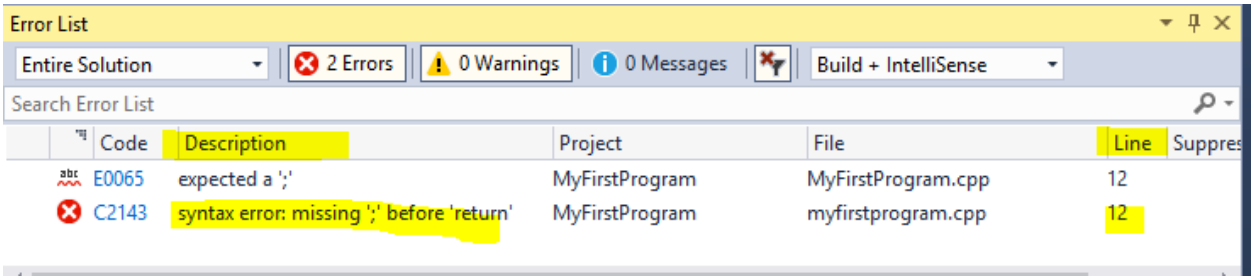
Αν το πρόγραμμα μας περιέχει συντακτικά λάθη θα εμφανιστεί το μήνυμα



Σε αυτή την περίπτωση πρέπει να **επιλέξουμε το No**. Σε περίπτωση που επιλέξουμε το Yes στην ουσία εκτελούμε την προηγούμενη έκδοση του προγράμματος μας που δεν είχε συντακτικά λάθη!!!

Τα λάθη τα συντακτικά που έχουμε εμφανίζονται στο παράθυρο output στο κάτω μέρος της οθόνης μαζί με μια πρόβλεψη που κάνει ο compiler σχετικά με το ποιο λάθος έχουμε κάνει και σε ποια γραμμή

είναι αυτό το λάθος (γενικά ο compiler πετυχαίνει στην πρόβλεψη για τον αριθμό γραμμής που περιέχει το λάθος, η πρόβλεψη για το ποιο είναι το λάθος δεν έχει τόσο μεγάλο ποσοστό επιτυχίας!).



Διορθώνουμε το συντακτικό μας λάθος και ξανατρέχουμε το πρόγραμμα.

Κλείσιμο ενός project

Όταν τελειώσουμε την επεξεργασία ενός project μπορούμε να το κλείσουμε επιλέγοντας **File->Close Solution**

Άσκηση 1

Να δημιουργήσετε ένα project με το όνομα Lab1_1 και στο οποίο να εισάγετε τον παρακάτω κώδικα. Προσπαθήστε να διορθώσετε τα συντακτικά λάθη και να εκτελέσετε το πρόγραμμα.

```
#include<iostream>
using namespace std;

int main() {
        int x;
        cout >> "Give me two integers";
        cin >> x >> y;
        cout << "the sum of the integers is " << x + y;
}
```

3. Βασικές λειτουργίες του Debugger στο Visual Studio

Μία από τις πολύ χρήσιμες λειτουργίες του MSVC είναι η δυνατότητα υποστήριξης debugging στον κώδικα που έχουμε δημιουργήσει. Με την χρήση του debugger μπορούμε να εκτελούμε εντολή προς εντολή τον κώδικά μας ενώ παράλληλα μπορούμε να βλέπουμε την «μνήμη» του προγράμματος δηλαδή τις τιμές που παίρνουν οι μεταβλητές και με αυτό τον τρόπο να διορθώσουμε τα προβλήματα που υπάρχουν. Η γνώση και χρήση του debugger είναι πολύ σημαντική για την εξέλιξη ενός προγραμματιστή.

Εκκίνηση / τερματισμός Debbuger

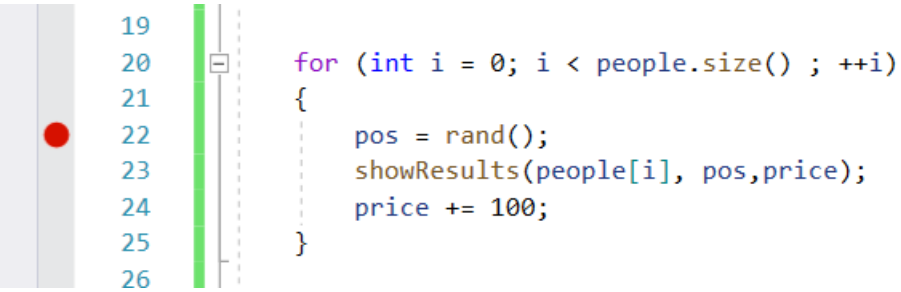
Δημιουργούμε ένα καινούργιο project στο MSVC με το όνομα **myDebugger** και προσθέτουμε τον παρακάτω κώδικα (Προσοχή: Σκοπός μας δεν είναι να καταλάβουμε την λειτουργία του κώδικα αλλά να γνωρίσουμε τις λειτουργίες που μας παρέχει ο debugger του MSVC)

```
#include<string>
#include<vector>
```

```
#include<algorithm>
#include<random>
#include <iostream>
#include<cstdlib>
#include<ctime>
using namespace std;

void showResults(const string& p, const int luck, const int price);

int main()
{
	vector<string> people{ "John","Nick","Peter","Anna" };
	int price = 100;
	int pos;
	srand(time(NULL));

	auto rng = default_random_engine{};
	shuffle(people.begin(), people.end(), rng);

	for (int i = 0; i < people.size(); ++i)
	{
		pos = rand();
		showResults(people[i], pos, price);
		price += 100;
	}

	return 0;
}


void showResults(const string& p, const int luck, const int price)
{
	int win;
	win = luck % 2;

	if (win == 0) {
		cout << p << " does not win " << price << " in euros" << endl;
	}
	else {
		cout << "Congrats " << p << " wins " << price << " in euros!" <<
endl;
	}
}
```

Εκτελούμε τον κώδικα από το μενού **Debug->Start debugging** (ή πλήκτρο **F5**). Σε περίπτωση που δεν έχουμε κάποιο συντακτικό λάθος, το πρόγραμμα εκτελείται αυτόματα και εμφανίζει μόνο τα αποτελέσματα εκτέλεσης του.

Ας προσθέσουμε ένα breakpoint για να «σταματήσουμε» το πρόγραμμα σε μία γραμμή του κώδικα. Η προσθήκη ενός breakpoint σε μια συγκεκριμένη γραμμή γίνεται κάνοντας κλικ στην γκρίζα στήλη αριστερά του αριθμού της γραμμής. Κάνοντας κλικ εμφανίζεται ένας κόκκινος κύκλος που υποδηλώνει ότι έχει τοποθετηθεί breakpoint σε αυτή την γραμμή, όπως φαίνεται και στην παρακάτω εικόνα. (Ξανακάνοντας κλικ πάνω στον κόκκινο κύκλο αφαιρείται)



Εκτελούμε και πάλι τον κώδικα πατώντας το **F5** (ή Debug->Start debugging). Η εκτέλεση του κώδικα θα σταματήσει **ΑΚΡΙΒΩΣ ΠΡΙΝ** την εκτέλεση της γραμμής κώδικα στον οποίο έχουμε τοποθετήσει το breakpoint (Αυτό φαίνεται και από ένα κίτρινο βέλος που δείχνει την γραμμή η οποία πρόκειται να εκτελεστεί στην συνέχεια)

Για να τερματίσουμε τον debugger μπορούμε να επιλέξουμε το **Debug->Stop Debugging** (**Shift+F5**). Εναλλακτικά μπορούμε να επανεκκινήσουμε τον debugger επιλέγοντας **Debug->Restart** (**Ctrl+Shift+F5**). Γενικότερα το Restart είναι πιο αποδοτικό από να τον τερματισμό και την επανεκκίνηση.

Μετακίνηση σε κώδικα

Ξεκινάμε την εκτέλεση του debugger σε περίπτωση που την είχαμε σταματήσει. Όπως είχαμε δει και προηγουμένως η εκτέλεση του κώδικα σταμάτησε στο breakpoint και ένα κίτρινο βέλος δείχνει την γραμμή που πρόκειται να εκτελεστεί στην συνέχεια. Οι επιλογές για την μετακίνηση μας μέσα στον κώδικα είναι οι παρακάτω:

- (**Continue F5**) Η εντολή **Debug-Continue** συνεχίζει την εκτέλεση του προγράμματος και σταματά στο επόμενο breakpoint (αν υπάρχει). Στο παράδειγμα μας όταν εκτελούμε τον debugger η εκτέλεση θα σταματήσει στο πρώτο breakpoint. Αν πατήσουμε F5 θα δούμε ότι δεν γίνεται τίποτα. Αυτό συμβαίνει γιατί το breakpoint είναι μέσα σε μία εντολή επανάληψης (for). Αυτό που έχει γίνει είναι ότι έχουν εκτελεστεί μια φορά οι εντολές επανάληψης και τώρα ήρθε πάλι στην εντολή με το breakpoint. Αυτό μπορούμε να το διαπιστώσουμε από την τιμή της μεταβλητής i που έχει γίνει ένα (το πως βλέπουμε την τιμή της θα το αναφέρουμε παρακάτω). Αν πατήσουμε το F5 άλλες τρεις φορές θα εκτελεστούν όλες οι επαναλήψεις του κώδικα και θα τερματιστεί η εκτέλεση του προγράμματος.
- (**Step Into F11**) Η εντολή **Debug-Step Into** πηγαίνει στην επόμενη εντολή του κώδικα. Στο παράδειγμα μας αν ξαναξεκινήσουμε τον debugger και πατήσουμε το F11 θα μας πάει στην εντολή που είναι η κλήση της συνάρτησης showResults. Αν ξαναπατήσουμε το F11 θα δούμε ότι η εκτέλεση θα μεταφερθεί μέσα στον κώδικα της showResults. Επομένως καταλαβαίνουμε ότι η Step Into μας μεταφέρει στην επόμενη εντολή που πρόκειται να εκτελεστεί. Μάλιστα αν η επόμενη εντολή είναι κλήση συνάρτησης μας μεταφέρει μέσα στον κώδικα της συνάρτησης. (Προσοχή! Μας μεταφέρει μέσα στον κώδικα της συνάρτησης μόνο αν την συνάρτηση αυτή την

έχουμε δημιουργήσει εμείς. Για παράδειγμα στην παραπάνω γραμμή δεν μας μεταφέρει μέσα στην rand που είναι συνάρτηση βιβλιοθήκης)

- (**Step Over F10**) Η εντολή **Debug-Step Over** πηγαίνει και αυτή στην επόμενη γραμμή του κώδικα. Η μεγάλη διαφορά με την Step Into είναι ότι η Step Over δεν πρόκειται να μπει στον κώδικα της συνάρτησης αν αυτή η επόμενη εντολή είναι η κλήση μιας συνάρτησης. Την Step Over την χρησιμοποιούμε αν γνωρίζουμε ότι μια συνάρτηση που καλείται είναι σωστή και δεν θέλουμε να εξετάσουμε τον κώδικα της. Ας ξαναξεκινήσουμε τον debugger. Επιλέγουμε F10 δύο φορές και παρατηρούμε ότι δεν πρόκειται η εκτέλεση να εισέλθει μέσα στον κώδικα της συνάρτησης showResults.
- (**Step Out Shift+F11**) Η εκτέλεση της εντολής Debug Step Out έχει νόημα αν η τρέχουσα εντολή που πρόκειται να εκτελεστεί είναι μέσα σε μία συνάρτηση και όχι στην main. Στην περίπτωση αυτή, αυτό που κάνει η εντολή Step Out είναι να εκτελέσει μονομιάς όλο τον κώδικα της συνάρτησης στην οποία βρισκόμαστε και να μεταφέρει την εκτέλεση στην εντολή αμέσως μετά την εντολή που κάλεσε την συνάρτηση. Για παρατηρήσουμε την λειτουργία της Step Out διαγράφουμε το break point που είχαμε βάλει και εισάγουμε ένα καινούργιο breakpoint στην πρώτη εντολή της συνάρτησης showResults. Στην συνέχεια επιλέγουμε Step Into (ή Step Over). Θα δούμε ότι θα μεταφέρει την εκτέλεση στην επόμενη εντολή της συνάρτησης showResults. Αν όμως επιλέξουμε την Step Out τότε θα εκτελεστούν όλες οι εντολές την συνάρτησης με μιας και ο έλεγχος θα μεταφερθεί στην συνάρτηση main στην εντολή που καλεί την συνάρτηση showResults.

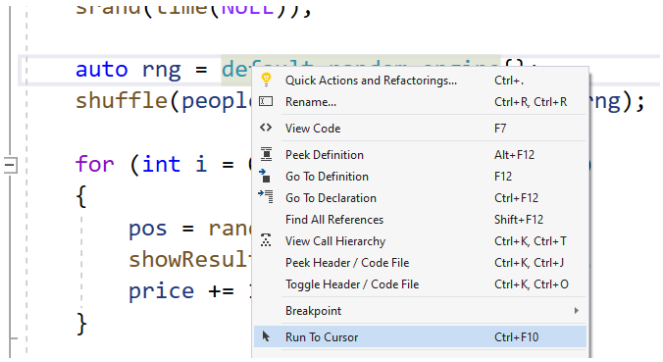
Δημιουργία σημείων ελέγχου

Δημιουργία breakpoints

Μέχρι στιγμή είδαμε ότι μπορούμε να σταματήσουμε την εκτέλεση του προγράμματος σε κάποια εντολή τοποθετώντας σε αυτή ένα breakpoint. Αυτό γίνεται πηγαίνοντας τον κέρσορα σε μία εντολή και επιλέγοντας Debug-Toggle BreakPoint. Αν επιλέξουμε την ίδια ενέργεια σε μια εντολή που έχει ήδη breakpoint αυτό θα αφαιρεθεί. Η ίδια ενέργεια μπορεί να γίνει με το να κάνουμε κλικ στην αριστερή γκρι λωρίδα πριν από κάθε μία εντολή.

Run to cursor

Αν όμως θέλουμε να δούμε στα γρήγορα αν μια εντολή εκτελείται κανονικά μπορούμε να κάνουμε το εξής. Κάνουμε δεξί κλικ σε μία εντολή την εκτέλεση της οποίας θέλουμε να παρατηρήσουμε και από το μενού επιλέγουμε run to cursor όπως φαίνεται και στην παρακάτω εικόνα



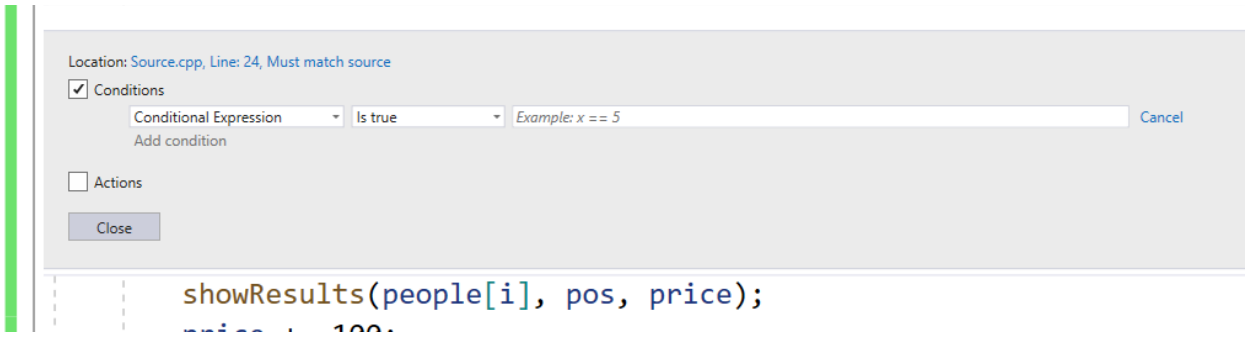
Το αποτέλεσμα είναι να εκτελεστεί ο debugger και να σταματήσει η εκτέλεση του προγράμματος μέχρι αυτή την εντολή σαν να είχαμε βάλει breakpoint. Να σημειώσουμε ότι την επόμενη φορά που θα ξανατρέξει ο debugger ΔΕΝ θα σταματήσει η εκτέλεση στην συγκεκριμένη εντολή.

Conditional breakpoints

Επίσης μπορούμε να τροποποιήσουμε τα breakpoints έτσι ώστε να μην σταματά η εκτέλεση πάντα αλλά μόνο όταν ισχύει μια συγκεκριμένη συνθήκη. Αυτή η λειτουργία ονομάζεται conditional breakpoins. Θα μετατρέψουμε ένα breakpoint σε conditional με τον εξής τρόπο. Αρχικά εισάγουμε, σε περίπτωση που το έχουμε διαγράψει, ένα breakpoint στην παρακάτω εντολή του κώδικα.

```
22      for (int i = 0; i < people.size(); ++i)
23      {
24          pos = rand();
25          showResults(people[i], pos, price);
26          price += 100;
27      }
```

Το παραπάνω breakpoint είναι μέσα σε μία εντολή for επομένως το πρόγραμμα μας θα σταματήσει για κάθε επανάληψη της for. Αν εμείς θέλουμε να σταματήσει μόνο την τελευταία φορά μπορούμε να κάνουμε δεξί κλικ στο εικονίδιο του breakpoint και να επιλέξουμε conditions. Θα εμφανιστεί η παρακάτω εικόνα.

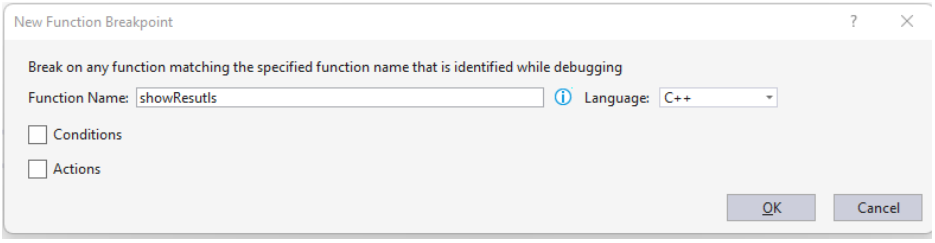


Οπότε μπορούμε να έρθουμε και να βάλουμε την συνθήκη που θέλουμε να ισχύει για να ενεργοποιηθεί το breakpoint σε αυτή την εντολή. Στο παράδειγμα μας βάζουμε την συνθήκη που υπάρχει στην παρακάτω εικόνα για να σταματήσουμε στην τελευταία επανάληψη της for. Εκτός από από συνθήκη σε ένα breakpoint μπορούμε να έχουμε

- ‘Ενα αριθμό στις οποίες θα περάσει η εκτέλεση του προγράμματος από το συγκεκριμένο breakpoint.
- Ένα μήνυμα που θα εμφανίζεται στο παράθυρο output και στην ουσία θα επιβεβαιώνει ότι η εκτέλεση του προγράμματος «πέρασε» από την συγκεκριμένη εντολή.

Function breakpoints

Ένας άλλος τρόπος δημιουργίας breakpoint είναι από το Debug-New Breakpoint-Function Breakpoint. Σε αυτή την περίπτωση εμφανίζεται ένα παράθυρο στο οποίο εισάγουμε το όνομα της συνάρτησης στην οποία θέλουμε να σταματήσει η εκτέλεση του προγράμματος. Στο παράδειγμα μας μπορούμε να γράψουμε το όνομα της showResults όπως φαίνεται στην παρακάτω εικόνα.

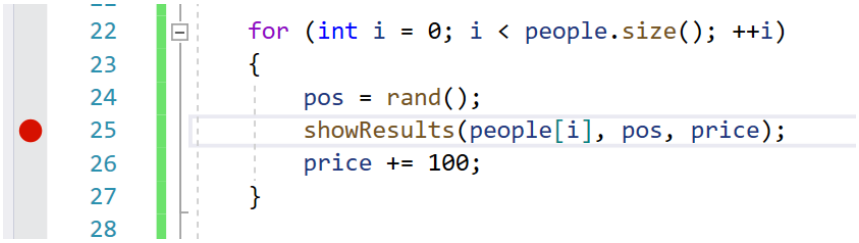


Με αποτέλεσμα την επόμενη φορά που θα εκτελέσουμε τον debugger η εκτέλεση θα σταματήσει στην πρώτη γραμμή της συνάρτησης showResults κάθε φορά που αυτή θα εκτελείται.

Έλεγχος τιμών μεταβλητών

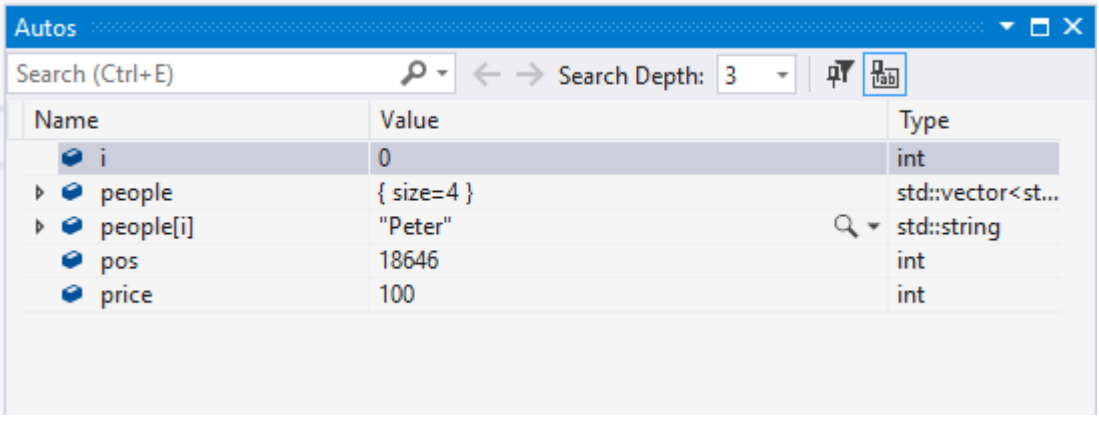
Ένα μεγάλο πλεονέκτημα που μας παρέχει ο debugger εκτός από το να βλέπουμε την εντολή προς εντολή εκτέλεση του προγράμματος μας είναι το γεγονός ότι εμφανίζει τις τιμές των μεταβλητών εκείνη την στιγμή.

Στο συγκεκριμένο παράδειγμα ας διαγράψουμε όλα τα breakpoints και ας εισάγουμε ένα στην εντολή που εμφανίζεται στην παρακάτω εικόνα.

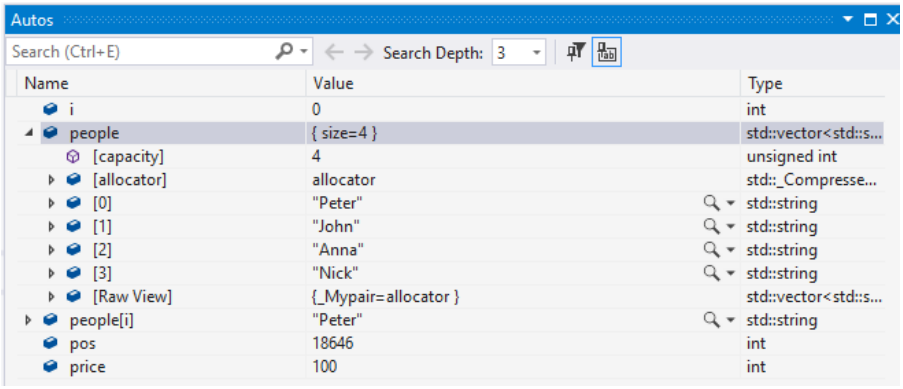


```
22    for (int i = 0; i < people.size(); ++i)
23    {
24        pos = rand();
25        showResults(people[i], pos, price);
26        price += 100;
27    }
28
```

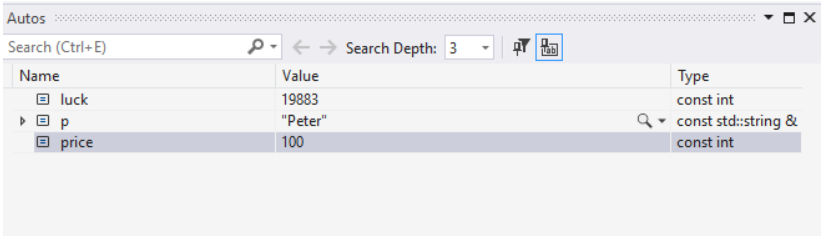
Εκτελούμε τον debugger και η εκτέλεση σταματά στην παραπάνω εντολή. Στο παράθυρο **Autos** (ακριβώς κάτω από το παράθυρο που εμφανίζεται ο κώδικας) φαίνονται οι τοπικές μεταβλητές που είναι δηλωμένες στον κώδικα και οι τιμές που έχουν αυτές οι μεταβλητές αυτή την στιγμή. Για παράδειγμα στον παραπάνω κώδικα την πρώτη φορά που θα μπει στο for loop η τιμή της μεταβλητής **i** θα είναι 0.



Επίσης και για τις μεταβλητές που είναι πιο «σύνθετου» τύπου μας εμφανίζει ένα βέλος μπροστά με το οποίο μπορούμε να δούμε ποιο αναλυτικά τις τιμές τους. Για παράδειγμα για την μεταβλητή **people** μας εμφανίζει την παρακάτω εικόνα.



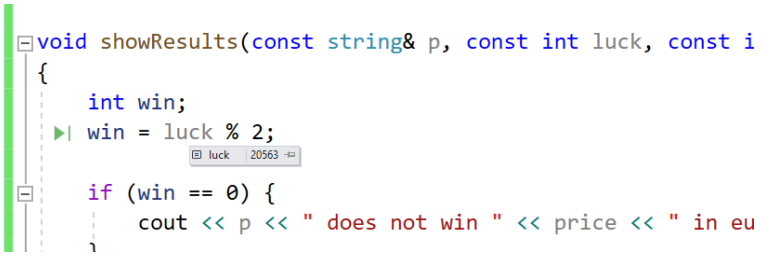
Επίσης αν πατήσουμε το πλήκτρο F11 (Step Into) θα εκτελεστεί η εντολή κλήση της συνάρτησης και η επόμενη εντολή θα είναι η πρώτη της συνάρτησης showResults. Εδώ βλέπουμε ότι το παράθυρο των μεταβλητών δείχνει ΜΟΝΟ τις μεταβλητές της συνάρτησης. Με αυτό τον τρόπο καταλαβαίνουμε ότι η συνάρτηση έχει πρόσβαση (scope) μόνο στις μεταβλητές που ορίζονται μέσα σε αυτή και στα input arguments όπως το είδαμε και στην θεωρία του μαθήματος.



Αν πατήσουμε τώρα το F11 και εκτελεστεί η εντολή ανάθεσης τιμής στην μεταβλητή win θα δούμε ότι αλλάζει η τιμής της και στο παράθυρο και εμφανίζεται μάλιστα με χρώμα κόκκινο. Με αυτό τον τρόπο καταλαβαίνουμε ποια μεταβλητή έχει επηρεαστεί από την εκτέλεση της εντολής.

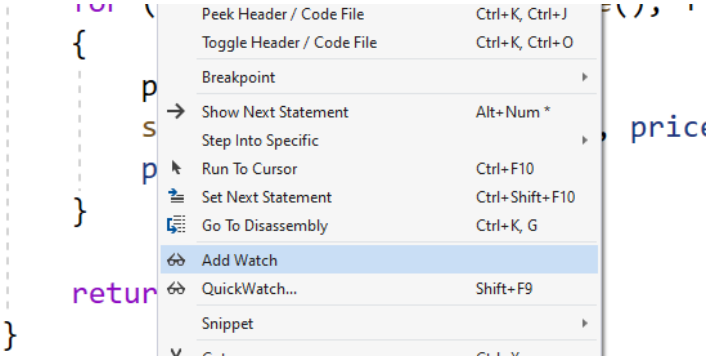
Με αυτόν το τρόπο έχουμε την δυνατότητα εκτέλεσης των εντολών του κώδικα με ταυτόχρονη προβολή των τιμών που παίρνουν οι μεταβλητές έτσι ώστε να γίνει κατανοητό που υπάρχει το λάθος στον κώδικα μας.

ΠΡΟΣΟΧΗ
Μπορούμε να δούμε την τιμή οποιασδήποτε μεταβλητής, η ακόμα και ολόκληρης έκφρασης, αν την επιλέξουμε με το ποντίκι κατά την διάρκεια που έχει σταματήσει η εκτέλεση του προγράμματος από τον Debugger σε ένα breakpoint



Add watch
Κατά την διάρκεια εκτέλεσης του debugger μπορεί να μας ενδιαφέρει η τιμή μιας η περισσοτέρων μεταβλητών και μόνο. Τότε μπορούμε να κάνουμε δεξί κλικ στο όνομα τους πάνω στον κώδικα και να

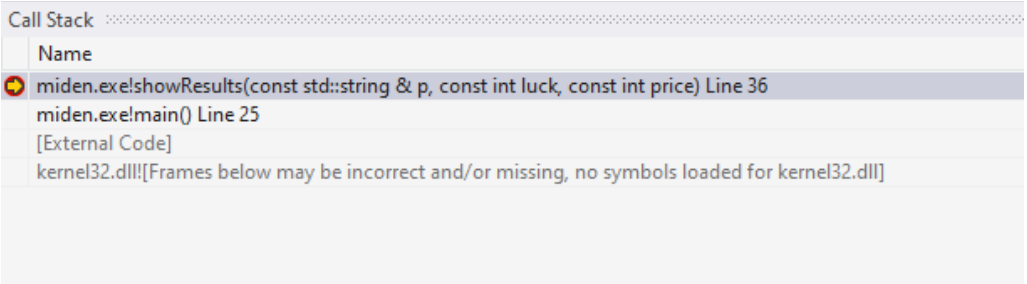
επιλέξουμε add watch. Με αυτό τον τρόπο θα παρουσιάζονται σε ένα παράθυρο συνέχεια μαζί με τις τιμές τους.



Έλεγχος call stack

Σε ένα πραγματικό σενάριο μπορεί να διαπιστώσουμε ότι υπάρχει πρόβλημα στον κώδικα μιας συνάρτησης και κατά συνέπεια εισάγουμε breakpoint σε αυτόν και εκτελούμε το debugger. Το πρόβλημα είναι ότι δεν γνωρίζουμε πως καλέστηκε αυτή η συνάρτηση και ποιες τιμές είχαν τα input argument. Αυτή την πληροφορία μας εμφανίζει το παράθυρο Call Stack.

Στο παράδειγμα μας εισάγουμε ένα breakpoint στην συνάρτηση μας και εκτελούμε τον debugger το παράθυρο callstack μας εμφανίζει ότι η κλήση έγινε από την συνάρτηση main στο αρχείο miden.exe καθώς και την γραμμή που υπάρχει η κλήση της συνάρτησης.



Άσκηση 2

Χρησιμοποιώντας τις λειτουργίες του debugger απαντήστε στις παρακάτω ερωτήσεις

1. Πόσες φορές θα εκτελεστεί η επανάληψη της for
2. Ποιες οι τιμές του price σε κάθε επανάληψη
3. Ποια είναι η λειτουργία της εντολής shuffle
4. Πόσες φορές θα καλεστεί η συνάρτηση showResults και ποια θα είναι τα ορίσματα εισόδου
5. Μπορούμε εκ των προτέρων να γνωρίζουμε πότε θα εκτελεστεί το if τμήμα και πότε το else τμήμα στην συνάρτηση;
6. Τι τιμές μπορεί να πάρει η μεταβλητή win
7. Μπορείτε να περιγράψετε με λίγα λόγια τι κάνει το πρόγραμμα

Άσκηση 3

Να δημιουργήσετε ένα καινούργιο project με το όνομα Lab0_2 και να εισάγετε το παρακάτω κώδικα

```
#include<iostream>
using namespace std;

void main() {
        int x,y;

        for (x = 0; x < 100; x += 5) {
                y = 2 * x - 1;
        }
}
```

Χρησιμοποιείστε τον debugger για να απαντήστε στις παρακάτω ερωτήσεις:

1. Πόσες φορές εκτελείται η for;
2. Ποιες είναι οι τιμές που παίρνει η x;
3. Ποιες είναι οι τιμές που παίρνει η y;