Aristotle Certification Training & Assessment

Τεχνοβλαστός Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης

Προγραμματισμός III

(προχωρημένο επίπεδο)

Αξιοποιώντας τον Προγραμματισμό

Έκδοση 1

ACTA ΑΕ – Τεχνοβλαστός Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης

Εγνατίας 1, 54630, Τηλ.: 2310 510 870, Fax: 2310 510 871, email: info@acta.edu.gr , www.acta.edu.gr

ΣΤΟΙΧΕΙΑ ΣΥΓΓΡΑΦΕΑ

Ονοματεπώνυμο Συγγραφέα: ΜΙΧΑΗΛ Ι. ΣΤΑΜΑΤΟΠΟΥΛΟΣ (Πληροφορικός ΠΕ19/ΠΕ86).

Είναι πτυχιούχος Πληροφορικός, του Τμήματος Πληροφορικής, της Σχολής Θετικών Επιστημών & Τεχνολογίας, Ε.Α.Π., Πατρών. Κατέχει παιδαγωγική και διδακτική επάρκεια στην εκπαίδευση της Πληροφορικής και έχει μεταπτυχιακή επιμόρφωση, σαν εκπαιδευτικός ειδικής αγωγής όπως επίσης και σαν εκπαιδευτής ενηλίκων, από το Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών (Ε.Κ.Π.Α.). Είναι επιστημονικός συνεργάτης – ερευνητής, στο Εργαστήριο Ευφυών Συστημάτων (i-Lab), του Τμήματος Πολιτισμικής Τεχνολογίας και Επικοινωνίας (Τ.Π.Τ.Ε.), της Σχολής Κοινωνικών Επιστημών, του Πανεπιστημίου Αιγαίου. Εργάζεται σαν μηχανικός λογισμικού και σαν εκπαιδευτικός Πληροφορικής για περισσότερα από 25 έτη και υπήρξε εκπρόσωπος στην Ελλάδα, κορυφαίων διεθνώς εταιρειών τεχνολογίας (FARO Technologies, CADZone, Trancite, A-T Solution).

Aristotle Certification Training & Assessment
Τεχνοβλαστός Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 2^{ο}

«ΟΡΓΑΝΩΜΕΝΕΣ ΜΟΡΦΕΣ ΔΕΔΟΜΕΝΩΝ»

ΚΕΦΑΛΑΙΟ 3^{ο}

«ΣΥΝΑΡΤΗΣΕΙΣ»

Aristotle Certification Training & Assessment
Τεχνοβλαστός Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης

ΚΕΦΑΛΑΙΟ 4ο

«ΕΙΔΙΚΑ ΘΕΜΑΤΑ ΤΗΣ C++»

Aristotle Certification Training & Assessment
Τεχνοβλαστός Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης

ΚΕΦΑΛΑΙΟ 5ο

«ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΕΙΑ – ΤΕΧΝΟΤΡΟΠΙΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ»

ΚΕΦΑΛΑΙΟ 6^{ο}

«ΕΡΓΑΣΤΗΡΙΑΚΗ ΜΕΛΕΤΗ»

Σκοπός και επιμέρους στόχοι

Προσδοκώμενα αποτελέσματα

6.1. Εκφώνηση εργασίας

6.2. Ενδεικτική επίλυση

Aristotle Certification Training & Assessment
Τεχνοβλαστός Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης

Γλωσσάριο σημαντικών όρων

Βιβλιογραφία

Επίλογος - Οδηγίες για περαιτέρω μελέτη

ΕΙΣΑΓΩΓΗ

Το σύγγραμμα αυτό αποτελεί την συνέχεια, του προηγούμενου συγγράμματος της σειράς, *Προγραμματισμός Ηλεκτρονικών Υπολογιστών & Μηχανών, της ACTA*. Έχοντας ήδη ολοκληρώσει, την παρουσίαση των βασικών τεχνικών στοιχείων της γλώσσας προγραμματισμού C++, η μελέτη συνεχίζεται με τον εμπλουτισμό των γνώσεων των εκπαιδευομένων σε ειδικότερα θέματα όπως οι, αλγοριθμικές δομές επανάληψης, οι οργανωμένες δομές δεδομένων, οι πίνακες, οι συναρτήσεις, αλλά και κάποια ιδιαίτερα θέματα όπως οι δείκτες μνήμης της C++ και η δημιουργία αναφορών σε μεταβλητές. Το σύγγραμμα περιλαμβάνει και ένα σημαντικό κεφάλαιο στο οποίο οι εκπαιδευόμενοι εισάγονται στο θέμα του αντικειμενοστραφούς προγραμματισμού (Object Oriented Programming). Ο σκοπός αυτού του κεφαλαίου είναι να ενημερωθούν επαρκώς οι εκπαιδευόμενοι με τις έννοιες της αντικειμενοστρέφειας και να έρθουν σε επαφή με τις ιδιαιτερότητες αυτής της σημαντικής τεχνοτροπίας.

Το σύγγραμμα, *Προγραμματισμός ΙΙΙ (προχωρημένο επίπεδο), Αξιοποιώντας τον Προγραμματισμό,* ολοκληρώνει την παρουσίαση όλων των απαραίτητων βασικών τεχνικών γνώσεων για την απόκτηση ικανοτήτων δημιουργίας προγραμμάτων στην γλώσσα C++.

1. ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ

Σκοπός και επιμέρους στόχοι

Ο σκοπός του πρώτου κεφαλαίου του συγγράμματος είναι η ολοκλήρωση της παρουσίασης των βασικών αλγοριθμικών δομών προγραμματισμού, με τις οποίες είναι δυνατό να καθοριστεί η ροή εκτέλεσης ενός προγράμματος. Ειδικότερα, στο κεφάλαιο παρουσιάζονται οι αλγοριθμικές δομές επανάληψης, με αντιπροσωπευτικά παραδείγματα, σχήματα και επεξηγηματικά σχόλια για την αποδοτική διαμόρφωση των εντολών, στην γλώσσα C++.

Προσδοκώμενα αποτελέσματα

Με την μελέτη του πρώτου κεφαλαίου οι εκπαιδευόμενοι θα μπορούν,

- να αντιλαμβάνονται και να περιγράφουν τις τρεις βασικές αλγοριθμικές δομές επανάληψης στις γλώσσες προγραμματισμού,
- να αντιλαμβάνονται τις διαφορές ανάμεσα στις δομές επανάληψης, με την συνθήκη στην αρχή, με την συνθήκη στο τέλος και με συνθήκη και βηματισμό,
- να αναφέρουν λόγους για τους οποίους θα αποφασίζουν ποια από τις τρεις βασικές αλγοριθμικές δομές επανάληψης, θα χρησιμοποιούν,
- να μπορούν να συντάξουν κάθε μια από τις τρεις αλγοριθμικές δομές επανάληψης, στην γλώσσα προγραμματισμού C++,
- να δημιουργούν τμήματα κώδικα C++, τα οποία θα μπορούν να επαναλαμβάνονται βάση, περιεχόμενου μεταβλητών, αποτελέσματος υπολογισμών και αποφάσεων λογικών εκφράσεων,
- να διακόπτουν την λειτουργία μιας δομής επανάληψης με κατάλληλες εντολές ανακατεύθυνσης,
- να ενσωματώνουν δομές επανάληψης μέσα σε άλλες δομές επανάληψης,
- να αντικαθιστούν την λειτουργία μιας αλγοριθμικής δομής επανάληψης με μια άλλη αλγοριθμική δομή επανάληψης.

Έννοιες – Λέξεις Κλειδιά

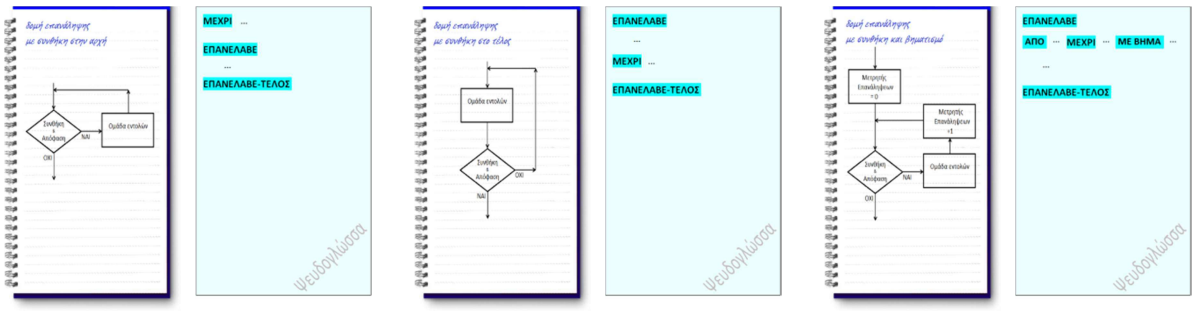
Επανάληψη με συνθήκη στην αρχή, Επανάληψη με συνθήκη στο τέλος, Επανάληψη με συνθήκη και βηματισμό, Απόφαση, Ανακύκλωση, Ατέρμονη ανακύκλωση, Εσωτερική δομή επανάληψης, Εξωτερική δομή επανάληψης, Εμφωλιασμένες δομές επανάληψης, Εντολές ανακατεύθυνσης, Εντολή while, Εντολή do ... while ..., Εντολή for.

Εισαγωγικές Παρατηρήσεις

Οι αλγοριθμικές δομές επανάληψης υπό συνθήκη, είναι οι ικανότερες δομές ελέγχου ροής για την γλώσσα προγραμματισμού C++. Οι δομές επανάληψης χρησιμοποιούνται σχεδόν σε όλες τις γλώσσες προγραμματισμού για να εκτελούν συγκεκριμένα τμήματα εντολών, μέχρι να ξεπεραστεί ένα προκαθορισμένο όριο επιθυμητών επαναλήψεων. Η γλώσσα C++, διαθέτει τρεις επιλογές διαμόρφωσης αλγοριθμικών δομών επανάληψης. Η διαφορά μεταξύ των τριών δομών, έγκειται κυρίως στον τρόπο με τον οποίο χειρίζονται τον τερματισμό του ορίου των επιθυμητών επαναλήψεων (στην αρχή, στο τέλος, ή με βηματισμό). Κάθε μια από τις αλγοριθμικές δομές είναι δυνατό να αντικατασταθεί με μια άλλη αλγοριθμική δομή και να έχει σε γενικές γραμμές την ίδια αποτελεσματικότητα. Είναι συνηθισμένη εκπαιδευτική διαδικασία και συχνό θέμα εξετάσεων, η αντικατάσταση μιας δομής επανάληψης με μια άλλη δομή επανάληψης.

Αλγοριθμικές Δομές Επανάληψης		
με την συνθήκη στην αρχή	με την συνθήκη στο τέλος	με συνθήκη και βηματισμό

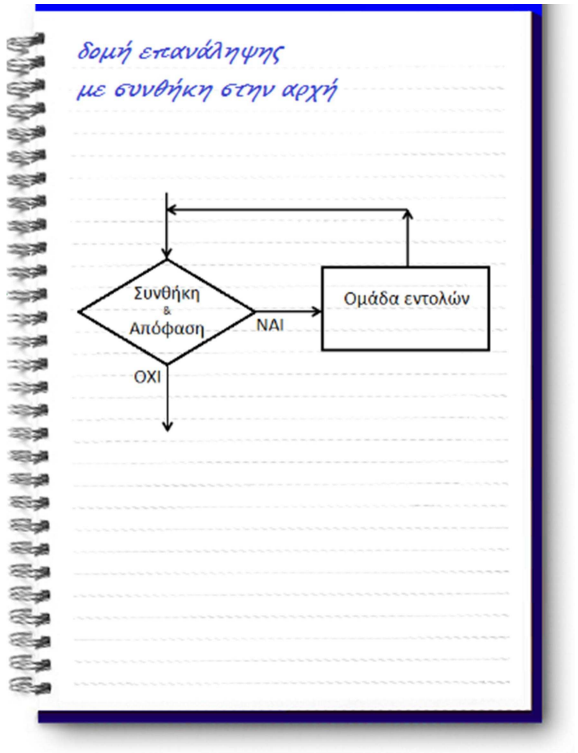
Οι δομές επανάληψης διευκολύνουν την δημιουργία ικανών, ευανάγνωστων και αποδοτικών προγραμμάτων. Οι έμπειροι προγραμματιστές, συνηθίζουν να χρησιμοποιούν την δομή επανάληψης, με συνθήκη και βηματισμό, όπως αυτή υλοποιείται στην C++, από την εντολή **for**.



Οι αλγοριθμικές δομές επανάληψης, αναλύονται στην συνέχεια με ολοκληρωμένα και λειτουργικά προγράμματα ενώ επικουρικά οι ίδιες δομές παρουσιάζονται σε λογικά διαγράμματα και σε ψευδογλώσσα.

1.1. Αλγοριθμική Δομή Επανάληψης με Συνθήκη στην Αρχή – while …

Η αλγοριθμική δομή επανάληψης υλοποιείται από έναν μηχανισμό ο οποίος μπορεί επαναλαμβανόμενα να εκτελεί το ίδιο κομμάτι εντολών προγραμματισμού, εγκλωβισμένων μέσα στον έλεγχο μιας συνθήκης. Η ελεγχόμενη συνθήκη μπορεί να είναι ένας απλός μετρητής επαναλήψεων ή να αποτελεί συνδυασμό πολλαπλών λογικών και αριθμητικών εκφράσεων, με τελικό ζητούμενο, μια οριστική απόφαση, ΑΛΗΘΕΣ ή ΨΕΥΔΕΣ (true ή false). Στην «αργκό» των προγραμματιστών, η εντολή ονομάζεται, *βρόγχος, ανακύκλωση* (loop), «*λούπα»,* ή *επανάληψη*, με την συνθήκη ελέγχου στην αρχή. Ακριβώς για τον λόγο αυτό η συνθήκη η οποία καθορίζεται με την δήλωση **while**, τοποθετείται στην αρχή της εντολής, ενώ οι καθαρές εκτελέσιμες εντολές ακολουθούν και οριοθετούνται από άγκιστρα **{ }**. Η εντολή αποτελεί μια ισχυρή δομή και χρησιμοποιείται ευρέως στον προγραμματισμό. Η συνθήκη ελέγχου είναι απολύτως δυναμική και σε κάθε ανακύκλωση η γλώσσα επανα-ελέγχει εάν έχει οριστική απόφαση, ΑΛΗΘΕΣ ή ΨΕΥΔΕΣ (true ή false), πράγμα που σημαίνει ότι ο προγραμματιστής έχει την ευχέρεια να ανα-καθορίζει, σε κάθε «λούπα» το περιεχόμενο των μεταβλητών που συμμετέχουν στην συνθήκη, με τον υψηλό όμως κίνδυνο η «λούπα», να πέσει σε ατέρμονη ανακύκλωση και να μην τελειώσει ποτέ. Καθώς ο έλεγχος της συνθήκης βρίσκεται στην αρχή, το σώμα των εντολών μπορεί να μην εκτελεστεί ποτέ.



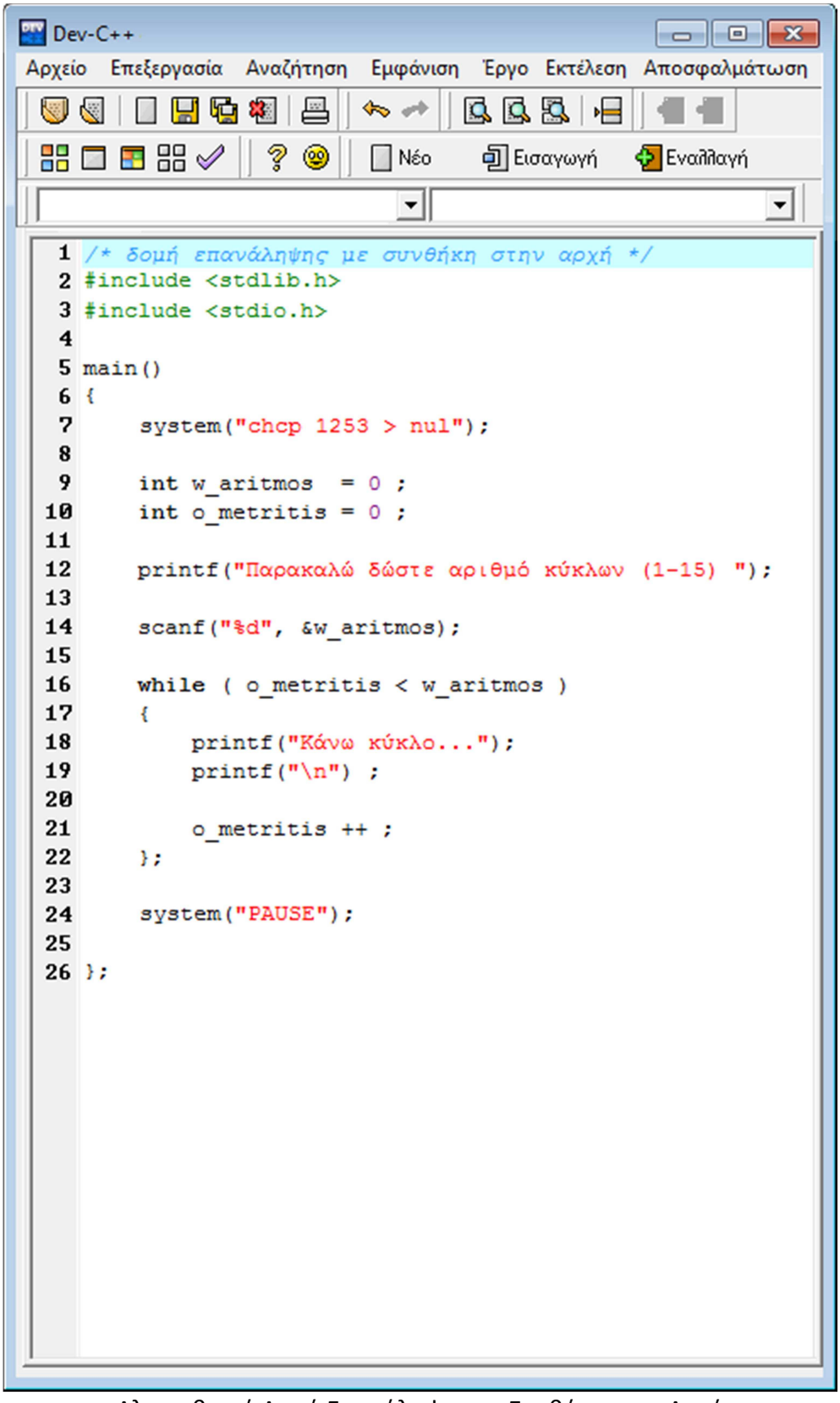
ΜΕΧΡΙ ...

ΕΠΑΝΕΛΑΒΕ

...

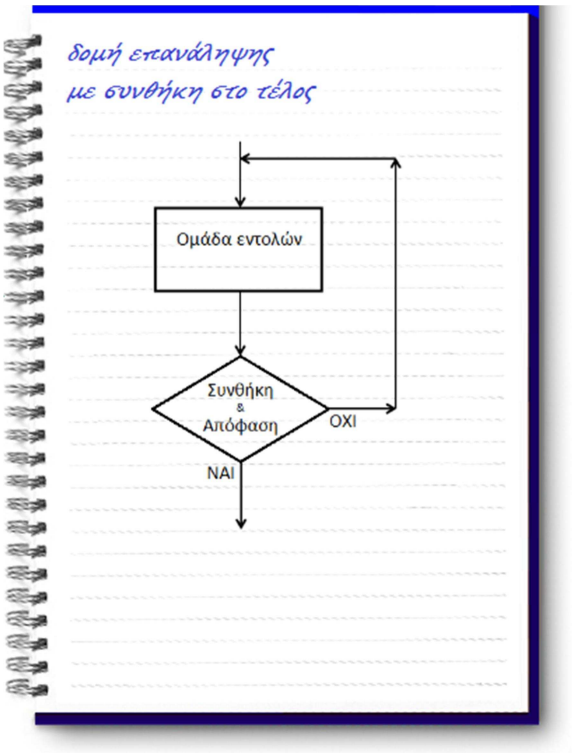
ΕΠΑΝΕΛΑΒΕ-ΤΕΛΟΣ

ψευδογλώσσα

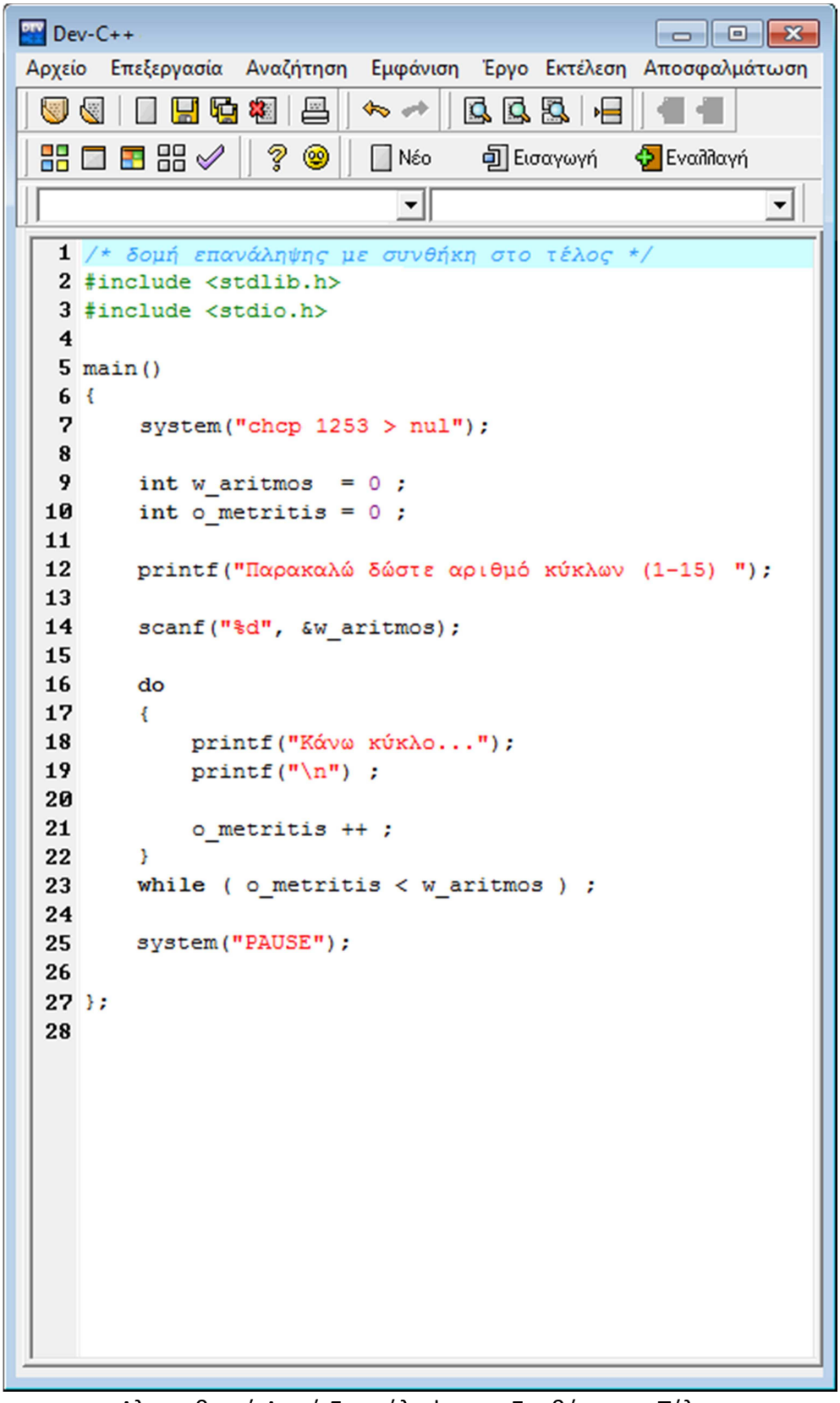


Αλγοριθμική Δομή Επανάληψης με Συνθήκη στην Αρχή

1.2. Αλγοριθμική Δομή Επανάληψης με Συνθήκη στο Τέλος – do … while



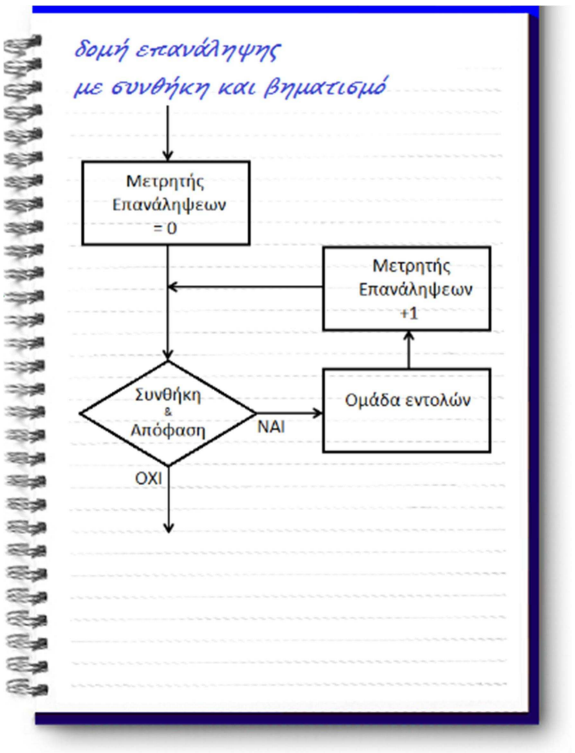
Η αλγοριθμική δομή επανάληψης με συνθήκη στο τέλος, της γλώσσας C++, υλοποιείται από έναν μηχανισμό ο οποίος μπορεί επαναλαμβανόμενα να εκτελεί το ίδιο κομμάτι εντολών προγραμματισμού, εγκλωβισμένων μέσα στον έλεγχο μιας συνθήκης, η οποία τοποθετείται στο τέλος της δομής. Η εντολή βασίζεται στις βασικές δηλώσεις **do … while** και καθώς ο έλεγχος βρίσκεται στο τέλος της εντολής, το σώμα των επαναλαμβανόμενων εντολών θα εκτελεστεί τουλάχιστον μια φορά. Η δομή ξεκινά με την δήλωση **do**, τα άγκιστρα **{ }**, μέσα στα οποία τοποθετούνται οι καθαρές επαναλαμβανόμενες εντολές και την δήλωση **while**, η οποία ακολουθεί και περιλαμβάνει την συνθήκη μέσα σε παρενθέσεις **( )**. Η ελεγχόμενη συνθήκη μπορεί να είναι ένας απλός μετρητής επαναλήψεων ή να αποτελεί συνδυασμό πολλαπλών λογικών και αριθμητικών εκφράσεων, με τελικό ζητούμενο, μια οριστική απόφαση, ΑΛΗΘΕΣ ή ΨΕΥΔΕΣ (true ή false). Η εντολή **do … while**, εντάσσεται στην ομάδα των ισχυρών αλγοριθμικών δομών με δυναμική συνθήκη ελέγχου και φυσικά τον κίνδυνο της ατέρμονης επανάληψης.

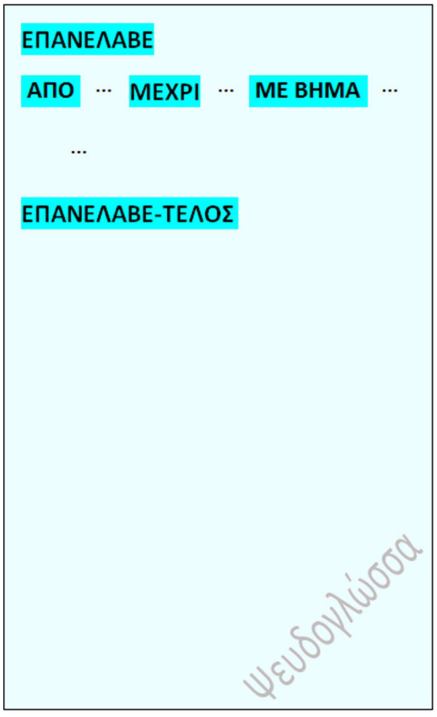


Αλγοριθμική Δομή Επανάληψης με Συνθήκη στο Τέλος

1.3. Αλγοριθμική Δομή Επανάληψης με Συνθήκη και Βηματισμό – for …

Από όλες τις δομές επανάληψης η ισχυρότερη, είναι η αλγοριθμική δομή επανάληψης με συνθήκη και βηματισμό. Η συγκεκριμένη λειτουργία, υλοποιείται στην C++, με την εντολή **for**, η οποία είναι ένας σύνθετος μηχανισμός που μπορεί επαναλαμβανόμενα, να εκτελεί ένα κομμάτι εντολών, εγκλωβισμένων μέσα στον έλεγχο μιας «έξυπνης» συνθήκης. Η «έξυπνη» αυτή συνθήκη μπορεί να είναι ένας απλός αυτόματος μετρητής επαναλήψεων ή να αποτελεί συνδυασμό πολλαπλών λογικών και αριθμητικών εκφράσεων, με τελικό ζητούμενο, μια οριστική απόφαση, ΑΛΗΘΕΣ ή ΨΕΥΔΕΣ (true ή false). Η εντολή **for**, χαρακτηρίζεται αυτόματη, καθώς εάν χρησιμοποιηθεί μετρητής επαναλήψεων, διατίθενται μέσα στις βασικές δηλώσεις της εντολής, δυο πρόσθετα στοιχεία, με τα οποία, α) μπορεί να αρχικοποιηθεί αυτόματα ο μετρητής επαναλήψεων και β) μπορεί να επηρεάζεται αυτόματα το περιεχόμενο του μετρητή επαναλήψεων.





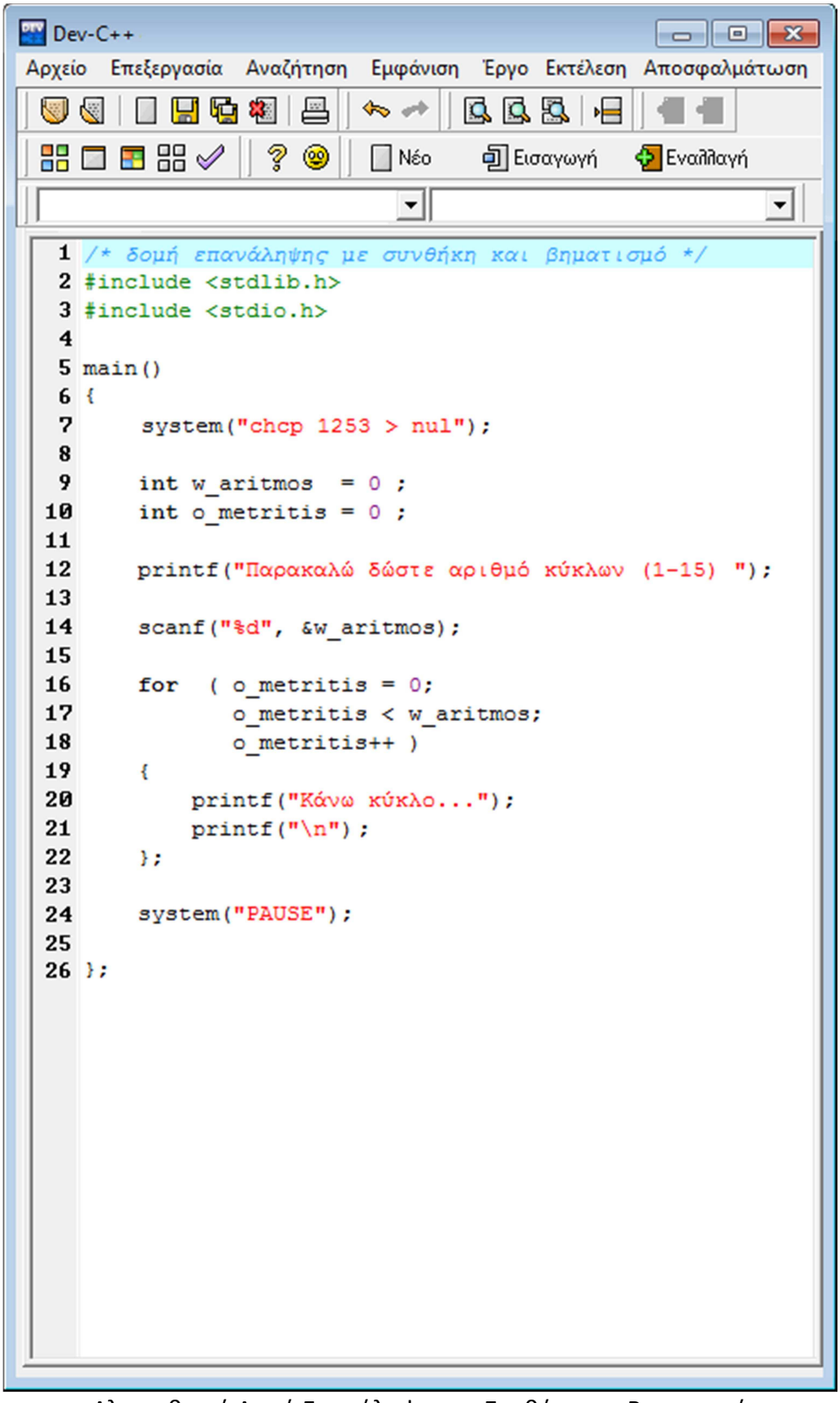
Η εντολή **for**, ονομάζεται επίσης, *βρόγχος, ανακύκλωση* (loop), «*λούπα»,* ή *επανάληψη*, με την «έξυπνη» συνθήκη ελέγχου να βρίσκεται στην αρχή. Καθώς ο έλεγχος της συνθήκης βρίσκεται στην αρχή, το σώμα των εντολών μπορεί να μην εκτελεστεί ποτέ. Η εντολή αποτελεί την

ισχυρότερη δομή επανάληψης της C++ και χρησιμοποιείται καθολικά από τους προγραμματιστές. Η γενική σύνταξη της εντολής είναι,

for (αρχικοποίηση_μετρητή,… ; έλεγχος_μετρητή,… ; βηματισμός μετρητή,…)

{ … } ;

Οι καθαρές εκτελέσιμες εντολές που θα επαναλαμβάνονται βρίσκονται κάτω από την δήλωση **for** και οριοθετούνται μέσα στα άγκιστρα **{ }**. Ολόκληρος όμως ο βασικός μηχανισμός λειτουργίας της **for**, βρίσκεται μέσα στις παρενθέσεις **( )** που συνοδεύουν την **for**. Συγκεκριμένα μέσα στην παρένθεση αυτή, οριοθετούνται τρεις ενότητες δηλώσεων, διαχωριζόμενες μεταξύ τους με τον χαρακτήρα semicolon **;**. Οι τρεις αυτές ενότητες αφορούν, την αρχικοποίηση, τον έλεγχο και τον βηματισμό μιας μεταβλητής που παίζει τον ρόλο του μετρητή επαναλήψεων. Η αρχικοποίηση του μετρητή γίνεται μόνο στην έναρξη λειτουργίας της εντολής, ενώ ο έλεγχος και ο βηματισμός, επαναλαμβάνονται σε κάθε ανακύκλωση. Ο βηματισμός του μετρητή επαναλήψεων συνηθίζεται να εκτελείται με μια δήλωση της μορφής, **o_metritis++**, με αυτό να σημαίνει ότι ο μετρητής σε κάθε ανακύκλωση θα αυξάνει την τιμή του κατά ένα. Ο βηματισμός μπορεί εκτός από θετικός, να είναι και αρνητικός με την δήλωση **o_metritis--**. Στην περίπτωση αυτή ο μετρητής σε κάθε ανακύκλωση θα μειώνει την τιμή του κατά ένα. Η δήλωση αρνητικού βηματισμού απαιτεί κατά την αρχικοποίηση του μετρητή, να αναθέτουμε επάνω του το άνω όριο των επαναλήψεων και όχι το μηδέν. Ένα πολύ λεπτό σημείο στην λειτουργία της εντολής **for**, είναι ότι, σε κάθε ανακύκλωση, ο έλεγχος του μετρητή επαναλήψεων γίνεται στην αρχή της ανακύκλωσης, ενώ ο βηματισμός του μετρητή επαναλήψεων (δηλαδή, η αύξηση ή η μείωση κατά ένα) γίνεται σαν τελευταία εντολή σε κάθε ανακύκλωση. Ολόκληρη η συνθήκη ελέγχου είναι απολύτως δυναμική και σε κάθε ανακύκλωση η γλώσσα επανα-ελέγχει εάν έχει οριστική απόφαση, ΑΛΗΘΕΣ ή ΨΕΥΔΕΣ (true ή false), πράγμα που σημαίνει ότι και εδώ ο προγραμματιστής έχει την ευχέρεια να ανα-καθορίζει, σε κάθε «λούπα» το περιεχόμενο των μεταβλητών που συμμετέχουν στην συνθήκη. Ταυτόχρονα όμως η εντολή σε κάθε ανακύκλωση εκτελεί και έναν δικό της αυτόματο έλεγχο, αφού μπορεί και επηρεάζει από μόνη της τον βηματισμό του μετρητή επαναλήψεων της και κατά συνέπεια να συμμετέχει αυτόματα και στον ανα-καθορισμό του περιεχομένου των μεταβλητών.



Αλγοριθμική Δομή Επανάληψης με Συνθήκη και Βηματισμό

1.4. Η εντολή for … στην σύνθετη μορφή της

Η εντολή **for**, εκτός από τον βασικό τρόπο λειτουργίας που ήδη περιγράφηκε, έχει την δυνατότητα να διαμορφωθεί με έναν ακόμα πιο σύνθετο τρόπο. Σε αυτή την εκτεταμένη μορφή, η σύνταξη της είναι,

```
for ( περιοχή_δηλώσεων_που_εκτελείται_στην_είσοδο,… ;
      περιοχή_δηλώσεων_1_που_εκτελείται_σε_κάθε_πέρασμα,… ;
      περιοχή_δηλώσεων_2_που_εκτελείται_σε_κάθε_πέρασμα,… )
        { … } ;
```

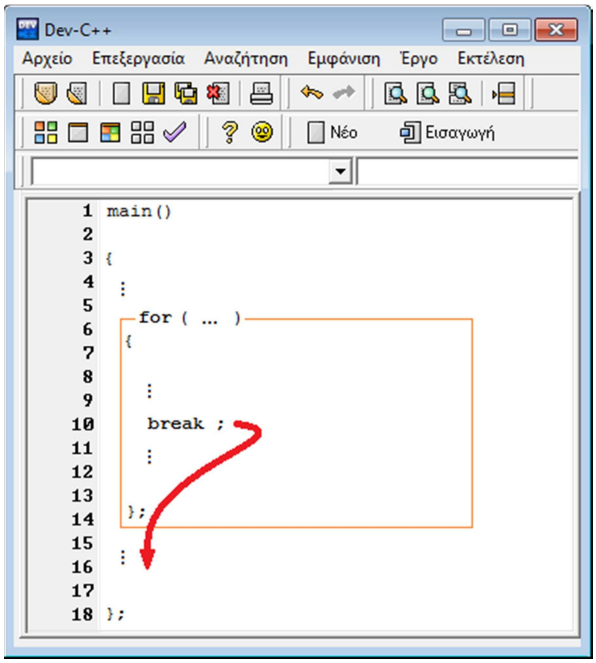
Με αυτή την σύνταξη η εντολή, μπορεί να εξαρτάται ταυτόχρονα από περισσότερες από μια μεταβλητές ελέγχου των ανακυκλώσεων που εκτελεί. Για παράδειγμα η επόμενη διαμόρφωση βασίζεται ταυτόχρονα σε τρεις μεταβλητές ελέγχου επαναλήψεων,

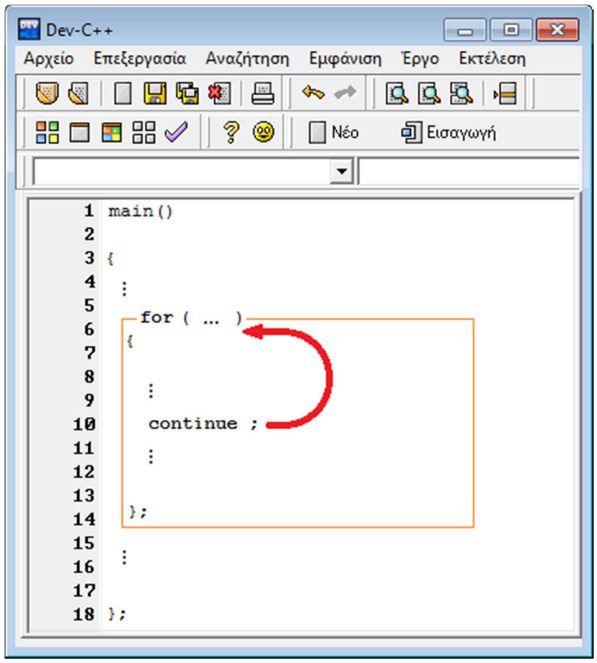
```
for ( int w_diktis_x=0, int w_diktis_y=0, int w_diktis_z=0 ;
      w_diktis_x+w_diktis_y+w_diktis_z<2000 ;
      w_diktis_x++, w_diktis_y++, w_diktis_z++ )
                              {
        printf( "w_diktis_x=%d w_diktis_y=%d w_diktis_z=%d \n", w_diktis_x,
                  w_diktis_y, w_diktis_z );
        };
```

όπου μέσα στην παρένθεση **( )**, η κάθε ενότητα δηλώσεων διαχωρίζεται από την επόμενη, με τον χαρακτήρα **;** που σημαίνει ότι η **for**, είναι στην πραγματικότητα ένας συνδυασμός εντολών. Επιπλέον, μέσα σε κάθε μια από αυτές τις τρεις περιοχές, οι μεταβλητές και οι πράξεις διαχωρίζονται μεταξύ τους με τον χαρακτήρα κόμμα **,** . Η εντολή σε αυτή την μορφή ανάπτυξης, αναδεικνύει τις ισχυρές δυνατότητες της γλώσσας. Κώδικες αυτού του επιπέδου δημιουργούν πολύ μικρά («συμπυκνωμένα») εκτελέσιμα αρχεία με αποτέλεσμα να παράγουν προγράμματα με εξαιρετική χωρική και χρονική πολυπλοκότητα και έτσι να δικαιολογούν τον λόγο για τον οποίο χιλιάδες προγραμματιστές προτιμούν και αγαπούν την γλώσσα C++.

1.5. Εντολές Ανακατεύθυνσης

Όλες οι αλγοριθμικές δομές επανάληψης, έχουν την δυνατότητα να συνδυαστούν με δυο πρόσθετες ειδικές εντολές. Συγκεκριμένα, η εντολή **for**, μπορεί μέσα στον κορμό των επαναλαμβανόμενων εντολών της που οριοθετούνται μέσα στα άγκιστρα **{ }**, να δεχτεί την ειδική εντολή ανακατεύθυνσης **break**. Η εντολή αυτή θα προκαλέσει την διακοπή λειτουργίας της τρέχουσας ανακύκλωσης, την απότομη διακοπή της λειτουργίας της **for** και τον «εξοστρακισμό» της «μπίλιας» ροής, έξω από την **for**, στην αμέσως επόμενη εντολή. Η εντολή **for**, μπορεί επίσης μέσα στον κορμό των επαναλαμβανόμενων εντολών της, να δεχτεί την εντολή ανακατεύθυνσης **continue**, που προκαλεί μια παρόμοια συμπεριφορά. Η ειδική εντολή **continue**, θα προκαλέσει την διακοπή λειτουργίας της τρέχουσας ανακύκλωσης και τον «εξοστρακισμό» της «μπίλιας» ροής, μέσα στην **for**, αλλά στην αμέσως επόμενη ανακύκλωση της. Στην εντολή **continue** δηλαδή, η **for** δεν σταματά την ατέρμονη ανακύκλωση της, αλλά συνεχίζει με την επόμενη επανάληψη.

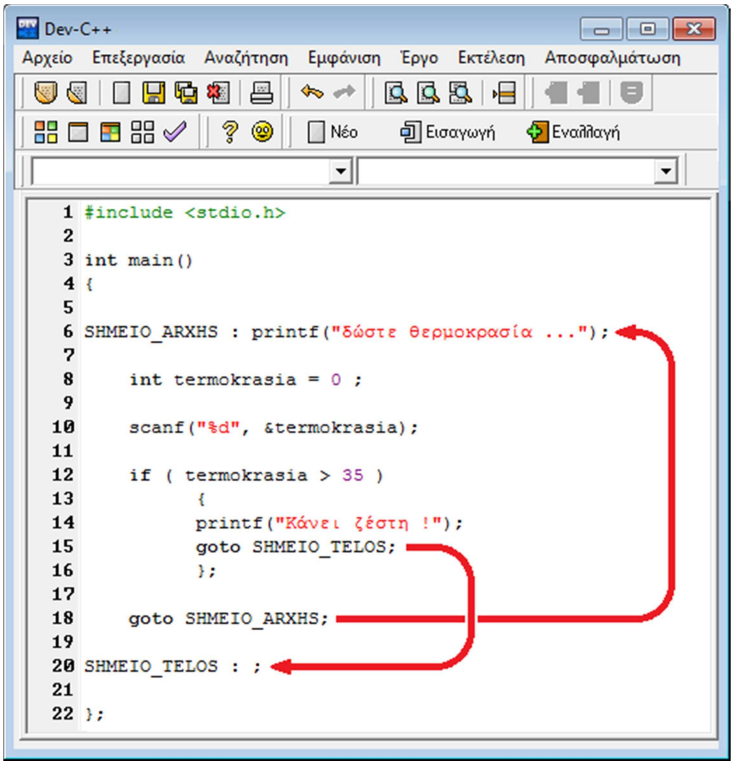




Οι εντολές ανακατεύθυνσης, **break** και **continue**, μπορούν να χρησιμοποιηθούν με τον ίδιο τρόπο και στις εντολές **while** και **do … while**.

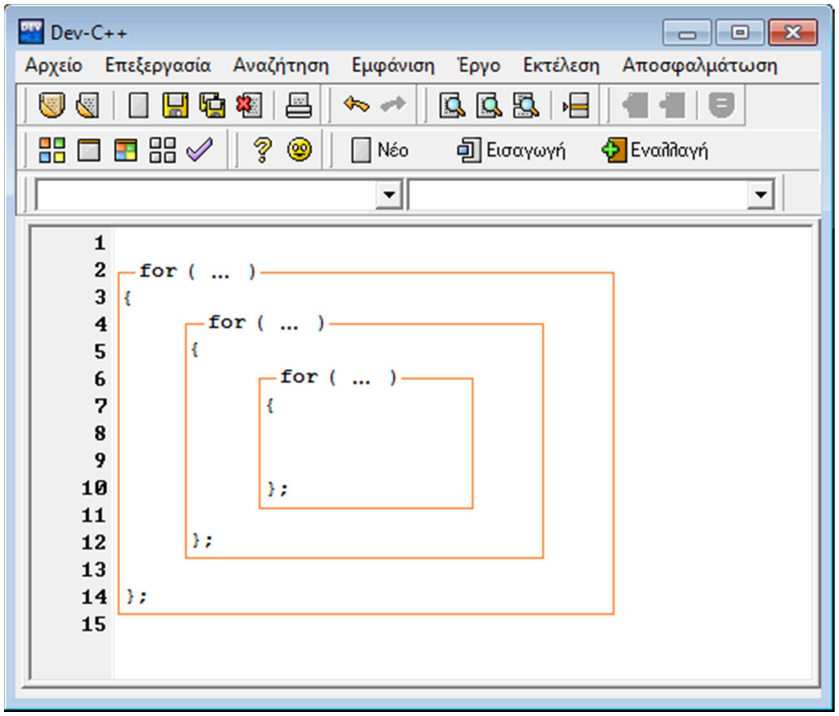
Εκτός από τις βασικές εντολές ανακατεύθυνσης, **break** και **continue**, η γλώσσα προγραμματισμού C++, διαθέτει και την εντολή ρητής ανακατεύθυνσης **goto**. Η εντολή **goto**, έχει μια ιδιαίτερη συμπεριφορά, καθώς δεν συνδυάζεται απαραίτητα με δομές επανάληψης ή με δομές επιλογής, αλλά μπορεί να συνδεθεί με σημεία ανακατεύθυνσης, οπουδήποτε μέσα σε ένα πρόγραμμα. Η εντολή **goto**, προκαλεί τον «εξοστρακισμό» της «μπίλιας» ροής, προς οποιοδήποτε σημείο μέσα σε ένα πρόγραμμα, στο οποίο του έχει δοθεί ένα συγκεκριμένο όνομα. Αυτά τα σημεία ανακατεύθυνσης, ονομάζονται ετικέτες (labels) και μπορούν να υπάρχουν σε πολλά διαφορετικά σημεία μέσα σε ένα πρόγραμμα. Επιπλέον, τα labels μπορούν να περιέχουν και εντολές.

Γενικότερα, η εντολή **goto**, εμφανίζεται σχεδόν σε όλες τις γλώσσες προγραμματισμού, χωρίς όμως να θεωρείται ιδιαίτερα δημοφιλής καθώς μπορεί εύκολα να προκαλέσει σύγχυση, εάν γίνεται κατάχρηση στην χρήση της. Ένα λεπτό σημείο στην εφαρμογή της είναι ότι δεν μπορούμε πραγματικά να ανακατευθύνουμε σε οποιοδήποτε σημείο. Για παράδειγμα, δεν μπορούμε να βάλουμε label, μέσα στο σώμα των εντολών μιας **while**, διότι έτσι ανακατευθύνεται η ροή μέσα σε μια δομή επανάληψης ο οποία δεν ξεκίνησε ποτέ και συνεπώς η γλώσσα δεν γνωρίζει πως να χειριστεί την έξοδο του **while**. Πάντως, η εντολή **goto** μπορεί πολλές φορές να λειτουργήσει «λυτρωτικά» και να βοηθήσει στην εύκολη και γρήγορη απεμπλοκή της ροής, μέσα από μια βαθιά εμφωλιασμένη κατάσταση.



1.6. Εμφωλιασμοί Αλγοριθμικών Δομών Επανάληψης

Οι αλγοριθμικές δομές επανάληψης, μπορούν να συνδυαστούν μεταξύ τους με τρόπο ώστε μια **for**, να βρίσκεται (να εμφωλιάζεται) μέσα σε μια άλλη **for**, ή μια **while** να εμφωλιάζεται μέσα σε μια άλλη **do … while**, κ.λπ. Η κατασκευή ενός αλγορίθμου με μια δομή επανάληψης να βρίσκεται μέσα σε μια άλλη δομή επανάληψης, σημαίνει ότι για κάθε μια ανακύκλωση της εξωτερικής δομής (εξωτερική «λούπα») θα εκτελεστούν όλες οι προβλεπόμενες ανακυκλώσεις της εσωτερικής δομής (εσωτερική «λούπα»). Μια τριπλή εμφωλιασμένη δομή επανάληψης, σημαίνει αντίστοιχα ότι, για κάθε μια ανακύκλωση της εξωτερικής δομής (εξωτερική «λούπα») θα εκτελεστούν μια προς μια όλες οι προβλεπόμενες ανακυκλώσεις της μεσαίας εσωτερικής δομής (μεσαία «λούπα») και για κάθε μια από αυτές τις μεσαίες ανακυκλώσεις, θα εκτελεστούν με την σειρά τους όλες οι προβλεπόμενες ανακυκλώσεις της εσωτερικής δομής (εσωτερική «λούπα»). Σε ένα πρόγραμμα C++, μπορούμε να έχουμε πολλαπλούς εμφωλιασμούς. Η διαδικασία συγγραφής προγραμμάτων με πολλαπλούς εμφωλιασμούς είναι μια συνηθισμένη πρακτική για τους προγραμματιστές.


Dev-C++
Αρχείο Επεξεργασία Αναζήτηση Εμφάνιση Έργο Εκτέλεση Αποσφαλμάτωση
Νέο
Εισαγωγή
Εναλλαγή
for (...)
{
for (...)
{
for (...)
{
};
};
};

Σύνοψη κεφαλαίου

Στο πρώτο κεφάλαιο ολοκληρώθηκε η μελέτη των βασικών αλγοριθμικών δομών της γλώσσας προγραμματισμού C++. Παρουσιάστηκαν αναλυτικά λειτουργικά παραδείγματα για όλες τις δομές επανάληψης και επεξηγήθηκαν ειδικά θέματα όπως οι ανακατευθύνσεις και οι εμφωλιασμοί δομών επανάληψης μέσα σε άλλες δομές.

Ερωτήσεις αξιολόγησης

Ερώτηση-1: Περιγράψτε, τι σημαίνει ο όρος, *αλγοριθμικές δομές επανάληψης*;

Ερώτηση-2: Στην αλγοριθμική δομή επανάληψης **do … while**, ο έλεγχος επανάληψης μπαίνει στην αρχή ή στο τέλος;

Ερώτηση-3: Στην αλγοριθμική δομή επανάληψης **while**, ο έλεγχος επανάληψης μπαίνει στην αρχή ή στο τέλος;

Ερώτηση-4: Στην αλγοριθμική δομή επανάληψης **for**, ο έλεγχος επανάληψης μπαίνει στην αρχή ή στο τέλος;

Ερώτηση-5: Στις αλγοριθμικές δομές επανάληψης, η ελεγχόμενη συνθήκη μπορεί να είναι ένας απλός μετρητής επαναλήψεων ή μπορεί να αποτελεί συνδυασμό πολλαπλών λογικών και αριθμητικών εκφράσεων;

Ερώτηση-6: Η αλγοριθμική δομή επανάληψης **for**, θεωρείται μια ισχυρή εντολή και ο βασικός μηχανισμός της βρίσκεται μέσα στις παρενθέσεις **( )** και τα τρία στοιχεία που τον συνοδεύουν. Στην απλή μορφή, ποια είναι αυτά τα τρία στοιχεία μέσα στις παρενθέσεις **( )**;

Ερώτηση-7: Στους εμφωλιασμούς δομών επανάληψης μπορούμε να συνδυάσουμε όλες τις δομές επανάληψης ή μόνο την εντολή **for**;

Ερώτηση-8: Περιγράψτε την σημασία της εντολής ανακατεύθυνσης **break**;

Ερώτηση-9: Περιγράψτε την σημασία της εντολής ανακατεύθυνσης **continue**;

Ερώτηση-10: Περιγράψτε την σημασία της εντολής ανακατεύθυνσης **goto**;

Ερώτηση-11: Είναι δυνατό να αντικατασταθεί μια αλγοριθμική δομή με μια άλλη αλγοριθμική δομή και να μην αλλάξει η αποτελεσματικότητα του προγράμματος;

Ερώτηση-12: Τι σημαίνει ατέρμονη επανάληψη, γιατί αποτελεί κίνδυνο και φόβο για τους προγραμματιστές;

Απαντήσεις ερωτήσεων αξιολόγησης

Απάντηση-1: Συμβουλευτείτε τις εισαγωγικές παρατηρήσεις του κεφαλαίου.

Απάντηση-2: Βρίσκεται στο τέλος.

Απάντηση-3: Βρίσκεται στην αρχή.

Απάντηση-4: Βρίσκεται στην αρχή.

Απάντηση-5: Μπορεί να είναι ένας απλός μετρητής επαναλήψεων ή μπορεί να είναι ένας συνδυασμός πολλαπλών λογικών και αριθμητικών εκφράσεων ή μπορεί να είναι και τα δύο;

Απάντηση-6: Στην απλή της μορφή, μέσα στην παρένθεση υπάρχουν 3 στοιχεία, η αρχικοποίηση, ο έλεγχος και ο βηματισμός μιας μεταβλητής που παίζει τον ρόλο του μετρητή επαναλήψεων.

Απάντηση-7: Μπορούμε να συνδυάσουμε τα πάντα.

Απάντηση-8: Συμβουλευτείτε την ενότητα 1.5. του κεφαλαίου.

Απάντηση-9: Συμβουλευτείτε την ενότητα 1.5. του κεφαλαίου.

Απάντηση-10: Συμβουλευτείτε την ενότητα 1.5. του κεφαλαίου.

Απάντηση-11: Ναι, είναι δυνατό.

Απάντηση-12: Σημαίνει, ότι μια δομή επανάληψης μπορεί να μην σταματήσει ποτέ λόγω κάποιου λογικού λάθους και ένα πρόγραμμα να «κολλήσει».

2. ΟΡΓΑΝΩΜΕΝΕΣ ΜΟΡΦΕΣ ΔΕΔΟΜΕΝΩΝ

Σκοπός και επιμέρους στόχοι

Ο σκοπός του δεύτερου κεφαλαίου του συγγράμματος είναι η παρουσίαση των θεμάτων τα οποία σχετίζονται με τις οργανωμένες δομές δεδομένων και ειδικότερα με τις δομές των πινάκων (διανύσματα). Στο κεφάλαιο γίνεται αρχικά μια συνοπτική περιγραφή για όλα τα απαραίτητα βασικά θέματα των οργανωμένων δομών δεδομένων και στην συνέχεια παρουσιάζεται μια εκτενής αναφορά για τους πίνακες στην γλώσσα προγραμματισμού C++. Το κεφάλαιο εμπλουτίζεται εκπαιδευτικά με πολλά επεξηγηματικά σχήματα καθώς και με ολοκληρωμένα αντιπροσωπευτικά παραδείγματα κώδικα, για την δημιουργία και την διαχείριση των πινάκων.

Προσδοκώμενα αποτελέσματα

Με την μελέτη του δεύτερου κεφαλαίου οι εκπαιδευόμενοι θα μπορούν,

- να αντιλαμβάνονται την διαφορά ανάμεσα στις πρωτογενείς μεταβλητές και τις οργανωμένες μορφές δεδομένων,
- να περιγράφουν την έννοια οργανωτικό σχήμα,
- να αναφέρουν τουλάχιστον 4 οργανωμένες μορφές δεδομένων,
- να περιγράφουν τι σημαίνει, γραμμική δομή δεδομένων και τι σημαίνει, μη γραμμική δομή δεδομένων,
- να περιγράφουν τις έννοιες, στατική δομή δεδομένων και δυναμική δομή δεδομένων,
- να ορίζουν την έννοια του πίνακα για τις γλώσσες προγραμματισμού,
- να περιγράψουν ένα παράδειγμα για την αναγκαία δημιουργία ενός πίνακα δεδομένων στην C++,
- να περιγράψουν τον λόγο χρήσης της ειδικής συνοδευτικής μεταβλητής, που ονομάζεται, δείκτης πίνακα (index),
- να αντιλαμβάνονται την διαφορά, ανάμεσα στον δείκτη του πίνακα (index) και του περιεχόμενο της θέσης που δείχνει ο δείκτης του πίνακα,
- να γνωρίζουν πως μπορούν να αναθέσουν τιμές σε έναν πίνακα της C++,

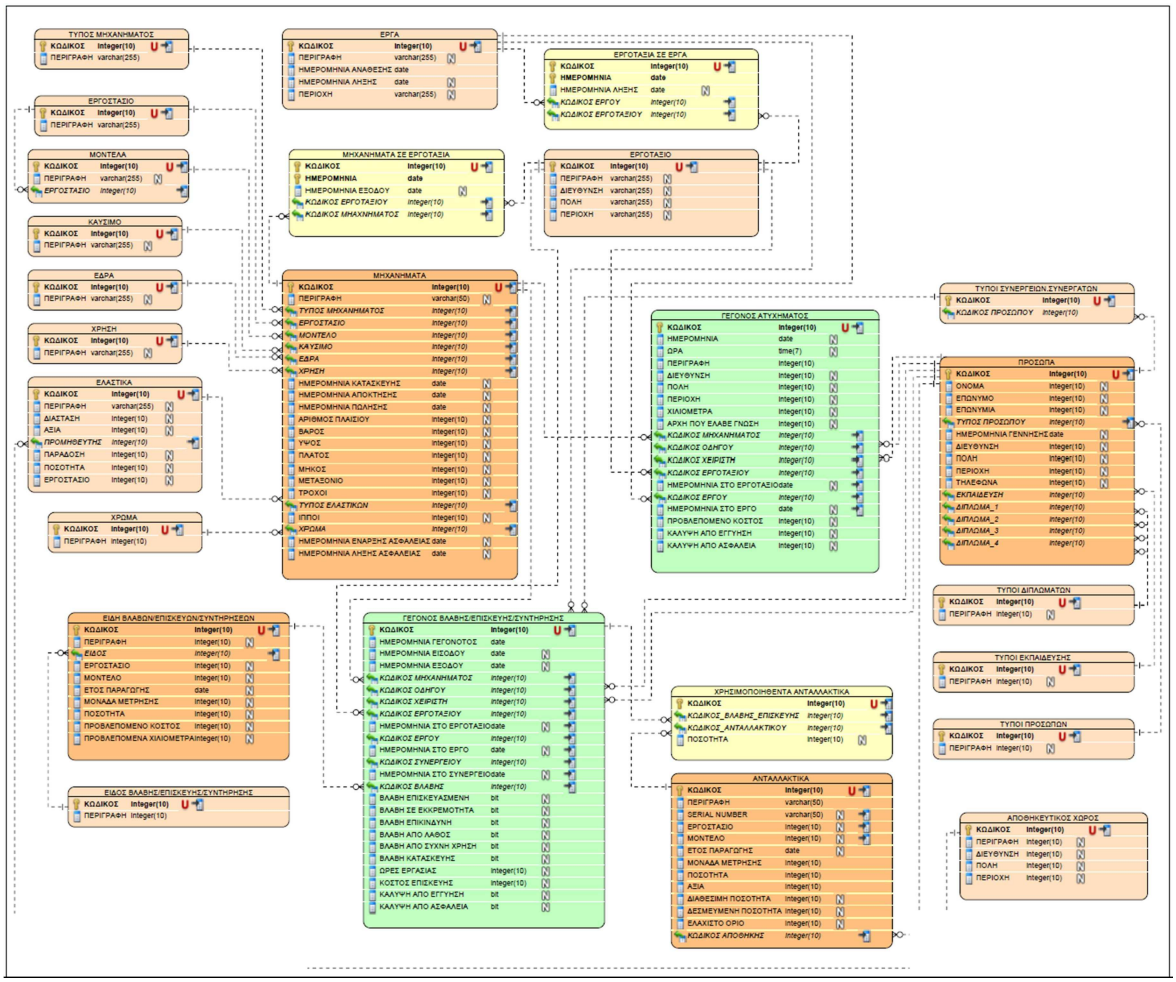
- να περιγράφουν πως μπορούν να προσεγγίσουν στοιχεία ενός πίνακα με την χρήση μιας μεταβλητής,
- να γνωρίζουν τον λόγο χρήσης της ειδικής συνοδευτικής μεταβλητής, που ονομάζεται, δείκτης πίνακα (index),
- να χρησιμοποιούν αλγοριθμικές δομές επανάληψης της C++, για να διαπερνούν πίνακες,
- να αντιλαμβάνονται την έννοια των πολλών διαστάσεων σε πίνακες, στην C++,
- να χειρίζονται αλφαριθμητικές μεταβλητές με πίνακες της C++.

Έννοιες – Λέξεις Κλειδιά

Δομημένες μορφές δεδομένων, Πίνακες, Εγγραφές, Λίστες, Δέντρα, Γράφοι, Αρχεία, Βάσεις δεδομένων, Οργανωτικό σχήμα, Γραμμικές δομές δεδομένων, Μη γραμμικές δομές δεδομένων, Στατικές δομές, Δυναμικές δομές, Διαπέραση πίνακα, Εισαγωγή στοιχείου, Διαγραφή στοιχείου, Αναδιάταξη πίνακα, Δείκτης πίνακα, Διατεταγμένος πίνακας, Μη διατεταγμένος πίνακας, Πίνακες πολλών διαστάσεων, Αλφαριθμητική μεταβλητή.

Εισαγωγικές Παρατηρήσεις

Η δημιουργία ενός προγράμματος σε μια γλώσσα προγραμματισμού δεν μπορεί να αφορά απλά και μόνο την περιγραφή αλγορίθμων που εκτελούν υπολογισμούς ή συγκρίνουν τα περιεχόμενα, από μεταβλητές που προκύπτουν από διεπαφές. Μια διεπαφή όσο σύνθετη και να είναι, δεν θα καταφέρει ποτέ να προμηθεύσει σε ένα πρόγραμμα τους όγκους των δεδομένων, στον βαθμό που ένας υπολογιστής έχει την ικανότητα να διαχειριστεί. Οι μεταβλητές άλλωστε, αποτελούν για μια γλώσσα προγραμματισμού, απλές θεμελιώδεις έννοιες που περιγράφουν ατομικές (atomic data types) ή πρωτογενείς δηλώσεις (int, float, char, κ.λπ.) για δέσμευση συγκεκριμένων ποσοτήτων μνήμης, ενώ οι υπολογιστές και κατά επέκταση οι γλώσσες προγραμματισμού έχουν κατασκευαστεί για να χειρίζονται μεγάλους όγκους δεδομένων, με μεγάλη ταχύτητα και χωρίς λάθη. Για όλους αυτούς τους λόγους στον προγραμματισμό των υπολογιστών τα δεδομένα προσδιορίζονται με μια περισσότερο σύνθετη έκφραση που ονομάζεται, *οργανωμένη μορφή δεδομένων* (structured data types).



Από την ίδια και μόνο την έκφραση είναι προφανές ότι, τα δεδομένα οργανώνονται (τακτοποιούνται, δομούνται, στοιχίζονται, ομαδοποιούνται, κ.λπ.), με κάποιο συγκεκριμένο οργανωτικό σχήμα. Το οργανωτικό αυτό σχήμα καθορίζει τον τρόπο με τον οποίο τα δεδομένα τακτοποιούνται μέσα στην μνήμη του υπολογιστή. Την οργάνωση αυτή, σύνθετη ή απλή, την καθορίζει ένας προγραμματιστής, ένας μηχανικός λογισμικού, ένας σχεδιαστής δεδομένων ή ακόμα και έναν οξυδερκής πελάτης ο οποίος παραγγέλνει σε έναν προγραμματιστή την κατασκευή ενός συγκεκριμένου λογισμικού. Οι οργανωμένες ή δομημένες μορφές δεδομένων ή απλά δομές δεδομένων (data structures), είναι πολλές και διαρκώς εξελίσσονται μέσα από την εξέλιξη των γλωσσών προγραμματισμού. Η έκταση του συγκεκριμένου γνωστικού αντικειμένου είναι εκτενής και στο κεφάλαιο αυτό προσεγγίζονται τα απαραίτητα στοιχεία του θέματος ώστε να είναι δυνατό οι εκπαιδευόμενοι να χειρίζονται κάποιες βασικές δομές δεδομένων.

2.1. Κατηγορίες Δομών Δεδομένων

Οι δομές δεδομένων παρουσιάζονται με πολλές και διαφορετικές μορφές, από τις ποιο στοιχειώδεις έως τις περισσότερο σύνθετες ή τις ανώτερες μορφές. Συνηθισμένες μορφές οργανωμένων δομών δεδομένων για τις γλώσσες προγραμματισμού είναι οι,

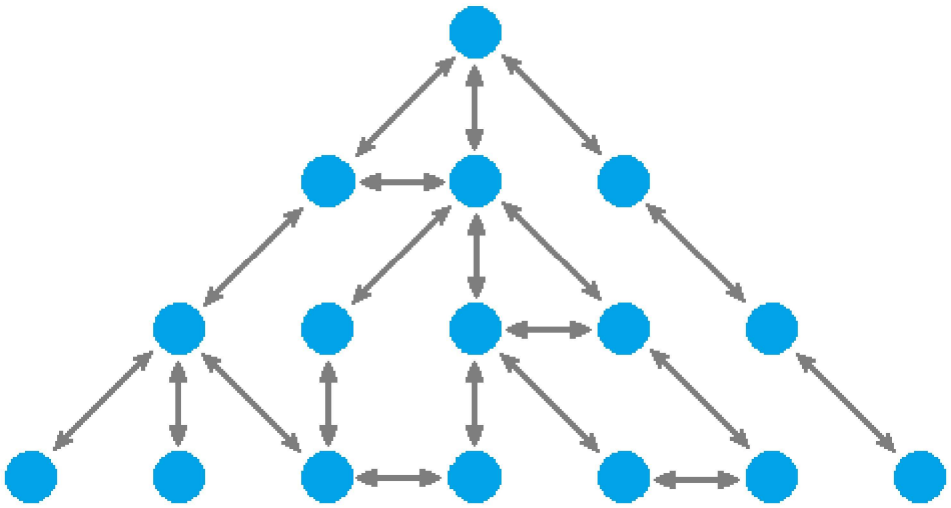
Δομές Δεδομένων
Οι πίνακες – Tables ή Arrays
Οι εγγραφές – Record
Οι γενικές λίστες (συνεχόμενες, συνδεδεμένες) – Linked lists
Οι ειδικές λίστες (στοίβες, ουρές) – Stacks
Τα δέντρα – Trees
Οι γράφοι (ή γραφήματα) – Graphs
Τα αρχεία – Files
Οι βάσεις δεδομένων – Data Bases

Μια ενδιάμεση μορφή η οποία βρίσκεται αμέσως πριν από όλες τις δομημένες μορφές δεδομένων αλλά μπορεί και αυτή να θεωρηθεί σαν μια μορφή οργάνωσης, είναι η δυνατότητα που παρέχει η C++, (όπως και πολλές άλλες γλώσσες προγραμματισμού) για την δημιουργία ενός σύνθετου τύπου μεταβλητής από τον συνδυασμό κάποιων άλλων στοιχειωδών τύπων

μεταβλητών. Το θέμα αυτό είναι σημαντικό για την γλώσσα C++, και θα το μελετήσουμε σε επόμενο κεφάλαιο.

Κάποιες από τις μορφές δομών δεδομένων ενσωματώνονται σαν βασικά δομικά στοιχεία στην C++, και κάποια άλλα θα πρέπει να δημιουργηθούν κατά περίπτωση από τον προγραμματιστή. Για παράδειγμα, οι πίνακες και οι εγγραφές είναι ενσωματωμένες δομές στις βασικές βιβλιοθήκες της γλώσσας ενώ οι υπόλοιπες δομές θα πρέπει να κατασκευαστούν κατά περίπτωση. Οι δομημένες μορφές δεδομένων διαχωρίζονται σε δυο σημαντικές κατηγορίες σε σχέση με τον τρόπο της οργάνωσης που εμπεριέχουν. Οι μορφές αυτές είναι, α) οι γραμμικές δομές δεδομένων και β) οι μη γραμμικές δομές δεδομένων.

Στην πρώτη μορφή, όλα τα στοιχεία της δομής βρίσκονται διασυνδεδεμένα σε μια μοναδική διάσταση, με το κάθε στοιχείο της δομής να συνδέεται γραμμικά με κάποιο άλλο στοιχείο, το οποίο ακολουθεί ή προηγείται.



Στην δεύτερη μορφή, τα στοιχεία μπορούν να διασυνδέονται ταυτόχρονα με πολλά άλλα στοιχεία, από και προς, πολλές διαφορετικές κατευθύνσεις, προσομοιώνοντας μια πολυδιάστατη σύνθετη κατασκευή διασύνδεσης. Οι δομές δεδομένων ανάλογα με τη μορφή τους, μπορούν μέσα σε ένα πρόγραμμα να παραμένουν αμετάβλητες ή να μεταβάλλονται σε

μέγεθος (σε πλήθος κόμβων). Στην πρώτη περίπτωση, ονομάζονται στατικές δομές (static data structures) και στην δεύτερη ονομάζονται δυναμικές δομές (dynamic data structures).

2.2. Τρόποι προσέγγισης Δομών Δεδομένων

Από την στιγμή που θα δημιουργηθεί μια δομή δεδομένων το αμέσως επόμενο θέμα το οποίο προκύπτει, είναι οι τρόπο με τους οποίους θα προσεγγίζεται η δομή. Με τον όρο *προσέγγιση,* εννοούμε τις πράξεις (ή τις ενέργειες) που θα χρειαστεί να γίνονται επάνω στο περιεχόμενο της δομής (στα δεδομένα), με κάποιες από αυτές τις πράξεις να αλλάζουν το περιεχόμενο της δομής και κάποιες να μην το αλλάζουν (static και dynamic data structures). Οι γλώσσες προγραμματισμού των υπολογιστών προσεγγίζουν τις δομές με τους επόμενους τρόπους,

Ενέργειες Δομών Δεδομένων
με διαδοχική προσέγγιση και επεξεργασία των στοιχείων της δομής – traversal
με αναζήτηση κάποιου συγκεκριμένου στοιχείου – search
με εισαγωγή (insert), ενός νέου στοιχείου – insert
με διαγραφή (delete), ενός στοιχείου από την δομή – delete
με αναδιάταξη (sort), των στοιχείων σε μια νέα σειρά – sort

Η δημιουργία μιας δομής δεδομένων και η εφαρμογή των μεθόδων προσέγγισης γίνεται με συγκεκριμένες εντολές προγραμματισμού που διαθέτουν όλες οι γλώσσες. Τα περισσότερα προγράμματα σε όλες τις γλώσσες προγραμματισμού και όλα τα συστήματα λογισμικού, είναι γεμάτα με αυτού του τύπου τις ενέργειες.

2.3. Οι Δομές Δεδομένων Πινάκων – Tables

Ίσως η σημαντικότερη δομή δεδομένων για όλες τις γλώσσες προγραμματισμού και όλα τα προγράμματα είναι οι πίνακες. Ο ορισμός *πίνακας*, μπορεί να περιγραφεί σαν, μια συγκέντρωση μεταβλητών ίδιου τύπου, οι οποίες ορίζονται με το ίδιο όνομα, αποθηκεύουν παρόμοιου τύπου δεδομένα και προσεγγίζονται μέσω της χρήσης ενός δείκτη σειράς. Για να γίνει αντιληπτή η σημασία των πινάκων και ο τρόπος λειτουργίας τους, θα χρησιμοποιήσουμε το παράδειγμα με τις θερμοκρασίες, το οποίο αναπτύχθηκε στο 1^{ο} σύγγραμμα της σειράς, *Προγραμματισμός Ηλεκτρονικών Υπολογιστών & Μηχανών*, της ΑΣΤΑ (κεφάλαιο 2, ενότητα 6).

Στο συγκεκριμένο παράδειγμα, 24 μετρήσεις θερμοκρασίας, χρησιμοποιήθηκαν από έναν αλγόριθμο για να υπολογιστεί η μέση θερμοκρασία μιας ημέρας.

Οι 24 θερμοκρασίες, αποτυπώνονταν από την ακολουθία των αριθμών, 16, 20, 22, 24, 25, 23, 20, 18, 16, 13, 12, 14, 10, 9, 8, 9, 11, 12, 10, 11, 9, 11, 12 και 14. Εάν ο ίδιος αλγόριθμος υλοποιείτο από ένα πρόγραμμα C++, θα χρειαζόταν να δημιουργηθούν 24 διαφορετικές μεταβλητές τύπου **int**, με ονόματα όπως, **w_temp_1**, **w_temp_2**, **w_temp_3** και σε κάθε μια από αυτές τις στοιχειώδεις μεταβλητές να τοποθετηθεί η αντίστοιχη θερμοκρασία. Αυτό θα μπορούσε να γίνει με απευθείας ανάθεση ή με ανάθεση μέσα από μια κατάλληλη διεπαφή. Εάν όμως, για κάποιον λόγο στην συνέχεια προέκυπτε η ανάγκη να διαφοροποιηθεί ο τρόπος υπολογισμού και να ανακατασκευαστεί το πρόγραμμα, έτσι ώστε να μπορεί διαχειριστεί θερμοκρασίες με δεκαδικά, θα έπρεπε να επέμβουμε σε κάθε μια από τις 24 μεταβλητές και να αντικαταστήσουμε όλες τις σχετικές δηλώσεις, **int** με **float**. Αυτή, και κάθε άλλη παρόμοια ενέργεια επαναλαμβανόμενης παρέμβασης στα προγράμματα, θεωρείται μια άχρηστη και κουραστική διαδικασία στον προγραμματισμό και μπορεί εύκολα να αποφευχθεί, εάν χρησιμοποιηθεί στην θέση των 24 μεταβλητών, ένας πίνακας. Οι πίνακες, είναι στοιχειώδεις μεταβλητές συγκεκριμένου τύπου, που έχουν για μέγεθος τις επαναλήψεις τους, που προκαθορίζονται βάση μιας συγκεκριμένης τιμής η οποία δηλώνεται την στιγμή που δημιουργείται ο πίνακας. Συνεπώς, η ανακατασκευή του προγράμματος για την διαχείριση θερμοκρασιών με δεκαδικά, θα απαιτούσε απλά και μόνο την διόρθωση αυτής της μοναδικής, στοιχειώδους μεταβλητής του πίνακα, από **int** σε **float**. Οι πίνακες προσφέρουν και πολλές άλλες ευκολίες, καθιστώντας τους ένα από τα ισχυρότερα εργαλεία προγραμματισμού. Τις ευκολίες που προσφέρουν οι πίνακες θα τις μελετήσουμε στην συνέχεια του κεφαλαίου. Στην γλώσσα C++, οι πίνακες ονομάζονται και διανύσματα (vectors). Η προσπέλαση των στοιχείων ενός πίνακα, γίνεται μέσω αναφοράς σε συγκεκριμένη θέση του πίνακα, όπως, διάβασε από την $5^{η}$ θέση του πίνακα, διάβασε από την $14^{η}$ θέση, την $24^{η}$ θέση, κ.λπ. Η προσπέλαση του κάθε στοιχείου, της θέσης δηλαδή του πίνακα, γίνεται με την χρήση μιας ειδικής συνοδευτικής μεταβλητής, που ονομάζεται, *δείκτης πίνακα* (index). Ένας πίνακας τελικά είναι διαδοχικές επαναλαμβανόμενες θέσεις μνήμης, μιας μεταβλητής συγκεκριμένου τύπου, που η κάθε θέση βρίσκεται η μια δίπλα στην άλλη και ενός δείκτη ο οποίος καταδεικνύει την θέση του κάθε στοιχείου που θέλουμε να προσεγγίσουμε. Η γενική μορφή της εντολής για την δημιουργία ενός πίνακα στην γλώσσα C++, είναι,

τύπος_μεταβλητής όνομα_πίνακα [θέσεις πίνακα] ;

Για παράδειγμα, η δήλωση,

```
int  w_o_pinakas_mou[7];
```

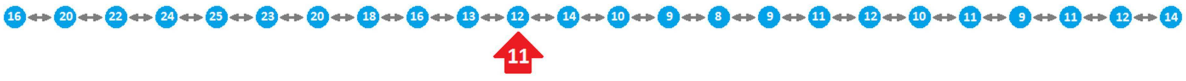
δημιουργεί στην C++, έναν πίνακα, σαν μια οργανωμένη δομή δεδομένων, γραμμικού τύπου, με το όνομα **w_o_pinakas_mou**, τύπου μεταβλητής **int** και με 7 διαθέσιμες θέσεις, για αποθήκευση ακέραιων αριθμών,

ενώ, η δήλωση,

```
int  w_temp_table[24];
```

δημιουργεί στην C++, έναν αντίστοιχο πίνακα με όνομα **w_temp_table** και 24 διαθέσιμες θέσεις, για την αποθήκευση των θερμοκρασιών,

τις οποίες εάν αναθέσουμε με τις κατάλληλες εντολές C++, η μνήμη του προγράμματος που έχει διατεθεί από τον υπολογιστή θα πάρει την επόμενη μορφή.



Στο σημείο που υποδεικνύει το κόκκινο βέλος βρίσκεται η 11^{η} θέση του πίνακα και η θέση αυτή περιέχει την θερμοκρασία, 12^{ο}. Εάν θέλαμε με μια εντολή C++, να αναφερθούμε σε αυτό

το στοιχείο (σε αυτή την θέση του πίνακα), θα μπορούσαμε να το κάνουμε με την επόμενη δήλωση,

```
w_temp_table[10];
```

και να παρατηρήσουμε ότι η θέση 11, που καταδεικνύουμε με το κόκκινο βέλος δεν αντιστοιχίζεται με τον αριθμό 11 αλλά με τον αριθμό 10, καθώς η γλώσσα C++, παρουσιάζει στο σημείο αυτό μια ιδιαιτερότητα καθώς δεν μετρά τις θέσεις των πινάκων, από το 1 (ένα) αλλά από το 0 (μηδέν). Εάν δηλαδή θέλουμε να αναφερθούμε στην τελευταία, την 24^{η} θέση του πίνακα με τις θερμοκρασίες, δεν θα δηλώσουμε, για δείκτη πίνακα τον αριθμό 24, αλλά τον αριθμό 23,


```
w_temp_table[23];
```

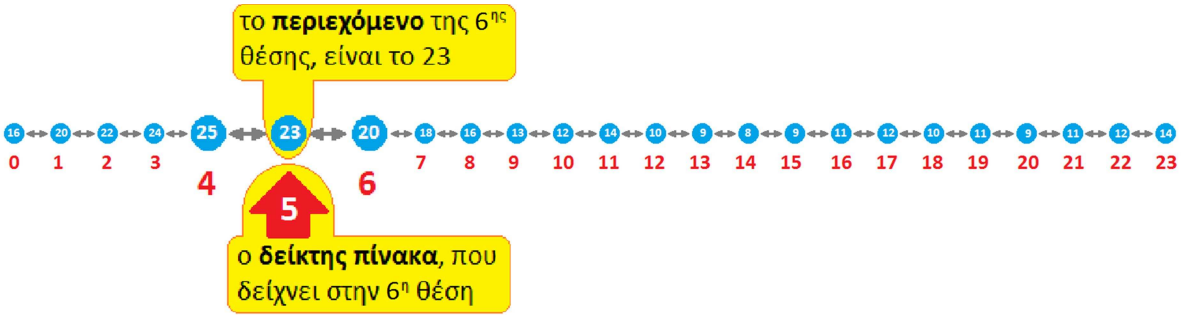
που με την σειρά του, εάν συνδυαστεί με μια εντολή **printf**, θα μπορούσε να γραφεί σαν,

```
printf("Η θερμοκρασία της τελευταίας θέσης είναι, %d", w_temp_table[23]) ;
```

με αποτέλεσμα, να εμφανίσει στην οθόνη, το κείμενο,

```
Η θερμοκρασία της τελευταίας θέσης είναι, 14
```

Ολοκληρώνοντας στην ενότητα αυτή την πρώτη παρουσίαση των οργανωμένων δομών δεδομένων, που στον προγραμματισμό αποκαλούνται, πίνακες, διανύσματα, tables, ή arrays οι εκπαιδευόμενοι θα πρέπει να συγκρατήσουν δυο βασικές έννοιες, οι οποίες είναι διαφορετικές αλλά συσχετίζονται άμεσα κατά την διαδικασία διαχείρισης ενός πίνακα,



- τον *δείκτη πίνακα* (index),
- και το *περιεχόμενο* της θέσης που δείχνει ο δείκτης του πίνακα.

Την ιδιαιτερότητα με τον δείκτη του πίνακα ο οποίος ξεκινά να μετρά από το 0 (μηδέν) και όχι από το 1 (ένα), παρουσιάζουν και όλες οι υπόλοιπες ομοιάζουσες γλώσσες με την C. Σε άλλες γλώσσες η μέτρηση των θέσεων των πινάκων, γίνεται κανονικά από την θέση 1 (ένα).

2.4. Ανάθεση αρχικών τιμών σε Πίνακες

Η ανάθεση αρχικών τιμών (το γέμισμα) μέσα στις θέσεις ενός πίνακα, είναι μια πολύ απλή αλλά βασική διαδικασία και απολύτως όμοια με την διαδικασία ανάθεσης αρχικών τιμών στις απλές στοιχειώδεις μεταβλητές. Η γενική μορφή σύνταξης της εντολής για την ανάθεση αρχικών τιμών σε έναν πίνακα στην γλώσσα C++, είναι,

```
τύπος_μεταβλητής  όνομα_πίνακα [θέσεις πίνακα] = { λίστα_τιμών } ;
```

με το τέλος της εντολής, να περιέχει ανάμεσα σε άγκιστρα **{ }** , την λίστα των τιμών, σαν μια παράθεση συμβατών τιμών (βάση του τύπου μεταβλητής του πίνακα), οι οποίες τιμές, διαχωρίζονται μεταξύ τους με κόμμα **,** .

Για παράδειγμα, η δημιουργία του πίνακα και η ανάθεση των 24 θερμοκρασιών, τύπου **int**, μπορεί να γίνει με την εντολή,

```
int   w_temp_table[24] = { 16, 20, 22, 24, 25, 23, 20, 18, 16, 13, 12, 14, 10, 9, 8, 9,
                           11, 12, 10, 11, 9, 11, 12, 14 } ;
```

η οποία θα προκαλέσει την δημιουργία της δομής,

που στην συνέχεια μπορούμε να προσεγγίσουμε τα 24 στοιχεία της, με την εντολή,

```
w_temp_table[θέση πίνακα];
```

όπου το **θέση πίνακα**, θα λαμβάνει τις ακέραιες τιμές, από το 0 μέχρι και το 23.
Στην περίπτωση που επιχειρήσουμε να προσεγγίσουμε μια θέση εκτός των δηλωμένων ορίων του πίνακα (για παράδειγμα, με την εντολή **w_temp_table[81]**), η C++, δεν θα μας σταματήσει στο compilation, το οποίο και θα δημιουργήσει κανονικά το εκτελέσιμο πρόγραμμα (**exe**), αλλά όταν θα τα εκτελεστεί το πρόγραμμα, ο δείκτης θα βγει εκτός των λογικών ορίων του πίνακα και θα αναγκάσει το πρόγραμμα να σταματήσει την λειτουργία του με μήνυμα λάθους. Αυτή η κατάσταση στην «αργκό» των προγραμματιστών, περιγράφεται σαν, «σκάσιμο» του πίνακα. Θα πρέπει δηλαδή να προσέχουμε να μην βγούμε εκτός των ορίων του πίνακα.

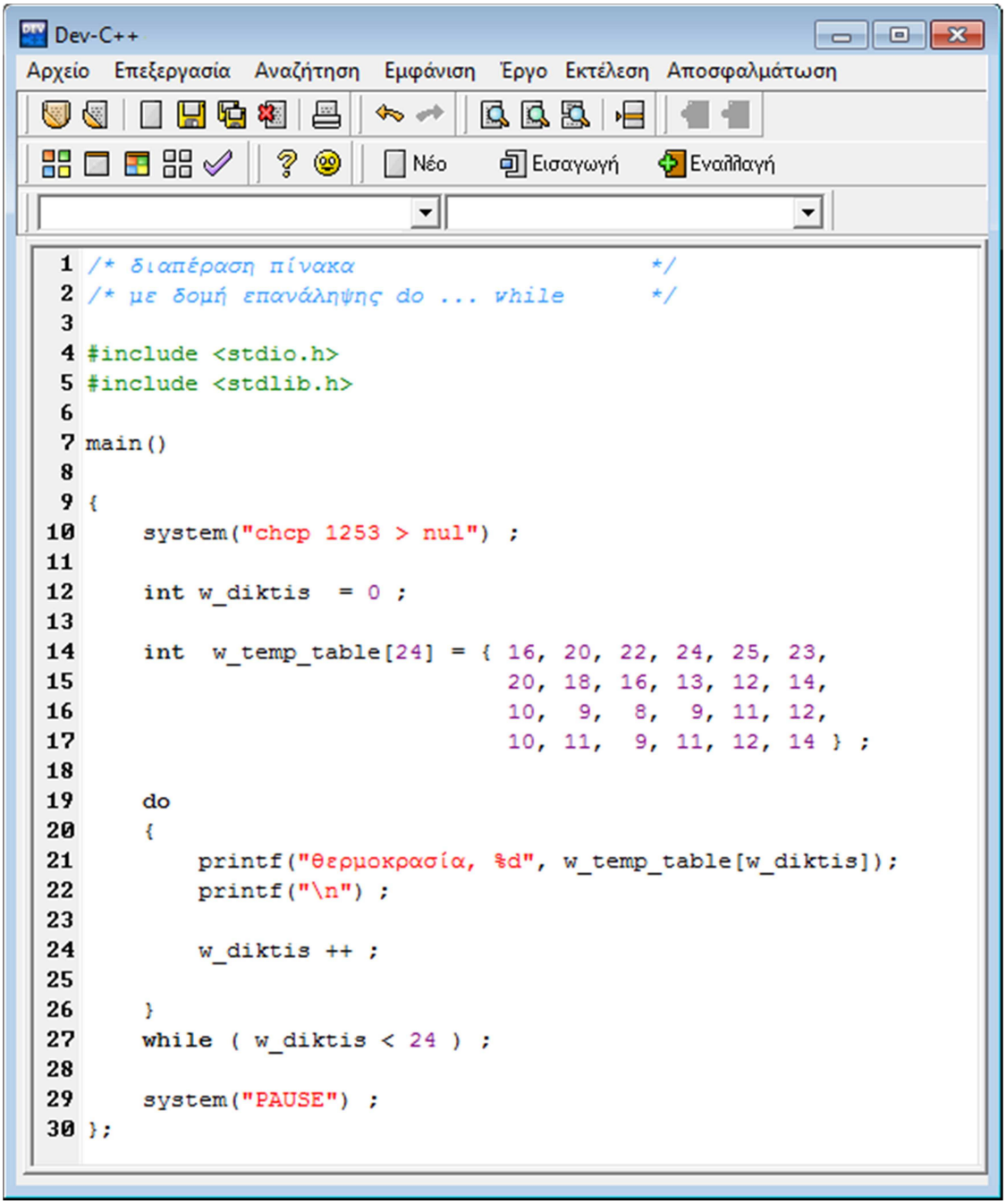
2.5. Προσέγγιση Στοιχείων Πίνακα με χρήση Μεταβλητής

Η προσέγγιση των στοιχείων ενός πίνακα δεν γίνεται μόνο με την ρητή αναφορά της θέσης του πίνακα (για παράδειγμα, με την εντολή **w_temp_table[11]**), αλλά μπορεί να χρησιμοποιηθεί στην θέση της ακέραιας αναφοράς (του αριθμού **11**), το όνομα μιας μεταβλητής. Η σύνταξη σε αυτή την περίπτωση γίνεται με την εντολή,

```
w_temp_table[όνομα_μεταβλητής] ;
```

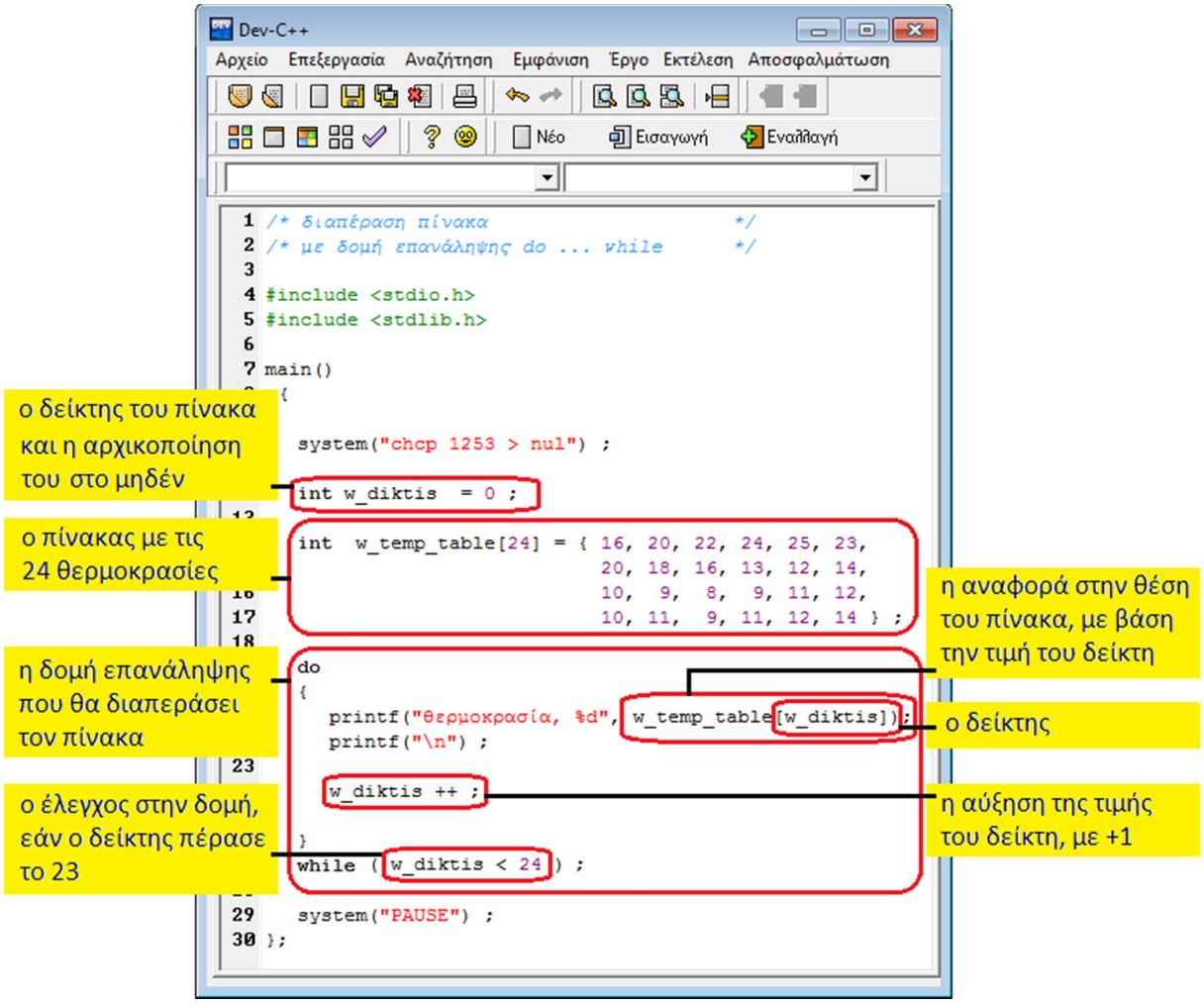
όπου το **όνομα_μεταβλητής**, θα πρέπει προηγουμένως να έχει δηλωθεί σαν μια ακέραια μεταβλητή τύπου **int**. Στην συνέχεια και με τις κατάλληλες εντολές ο πίνακας **w_temp_table** μπορεί να προσεγγίζεται με την χρήση αυτής της μεταβλητής και των τιμών που μπορεί να λαμβάνει (τις ακέραιες τιμές, από το 0 μέχρι και το 23). Η προσέγγιση των στοιχείων ενός πίνακα με χρήση μεταβλητής και σε συνδυασμό με μια αλγοριθμική δομή επανάληψης, αποτελεί μια ιδιαίτερα ισχυρή τεχνική προγραμματισμού για την αυτόματη διαδοχική

προσπέλαση των στοιχείων ενός πίνακα. Η τεχνική υλοποιείται με μια δομή επανάληψης και την επαναχρησιμοποίηση της μεταβλητής επαναλήψεων της δομής, σαν δείκτη του πίνακα. Καθώς η δομή επανάληψης ανακυκλώνεται στην βάση του μετρητή επαναλήψεων της, μια εντολή αναφοράς στον πίνακα με δείκτη αυτή την ίδια μεταβλητή, η οποία εμφωλιάζεται μέσα στην επανάληψη, έχει σαν αποτέλεσμα την διαδοχική προσέγγιση των στοιχείων του πίνακα. Η τεχνική παρουσιάζεται στην συνέχεια με δυο ολοκληρωμένα παραδείγματα C++, ένα με την εντολή επανάληψης **do…while** και ένα με την εντολή **for…**. Οι δυο κώδικες, προσεγγίζουν τον πίνακα των θερμοκρασιών, τον διαπερνούν και τυπώνουν στην οθόνη διαδοχικά τις 24 θερμοκρασίες. Στην πρώτη περίπτωση της **do…while**, ακολουθεί υποβοηθητικός σχολιασμός ενώ στην δεύτερη περίπτωση της **for…**, καλούνται οι εκπαιδευόμενοι να τοποθετήσουν κάθε μια από 7 κίτρινες ετικέτες (με τα σχόλια) στα σημεία του κώδικα που κατά την γνώμη τους η εντολή **for …**, τα υλοποιεί.

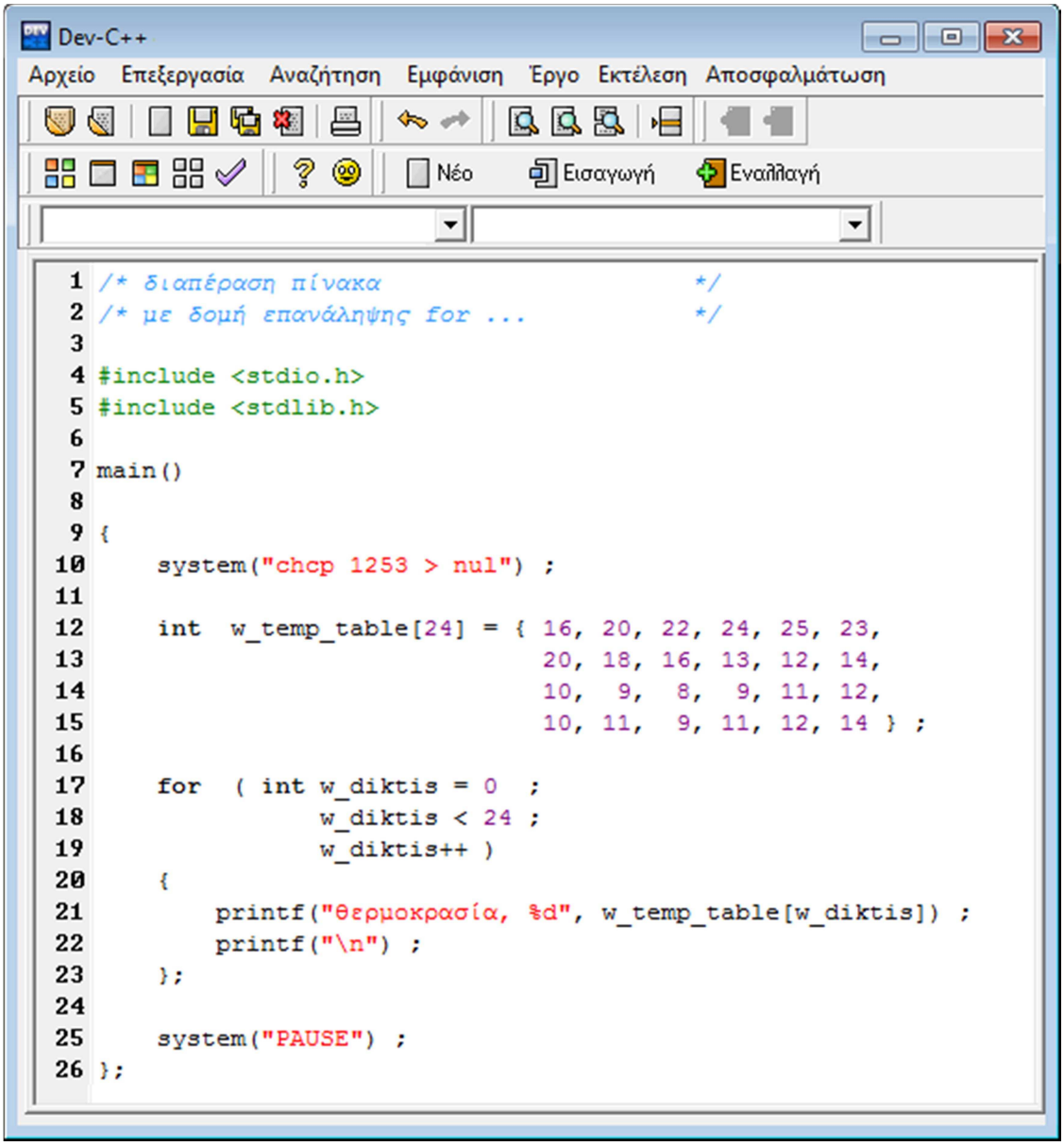



```c
/* διαπέραση πίνακα                    */
/* με δομή επανάληψης do ... while     */

#include <stdio.h>
#include <stdlib.h>

main()

{
    system("chcp 1253 > nul") ;

    int w_diktis  = 0 ;

    int  w_temp_table[24] = { 16, 20, 22, 24, 25, 23,
                              20, 18, 16, 13, 12, 14,
                              10,  9,  8,  9, 11, 12,
                              10, 11,  9, 11, 12, 14 } ;

    do
    {
        printf("θερμοκρασία, %d", w_temp_table[w_diktis]);
        printf("\n") ;

        w_diktis ++ ;

    }
    while ( w_diktis < 24 ) ;

    system("PAUSE") ;
};
```

Η προσέγγιση του πίνακα με την εντολή **do…while**

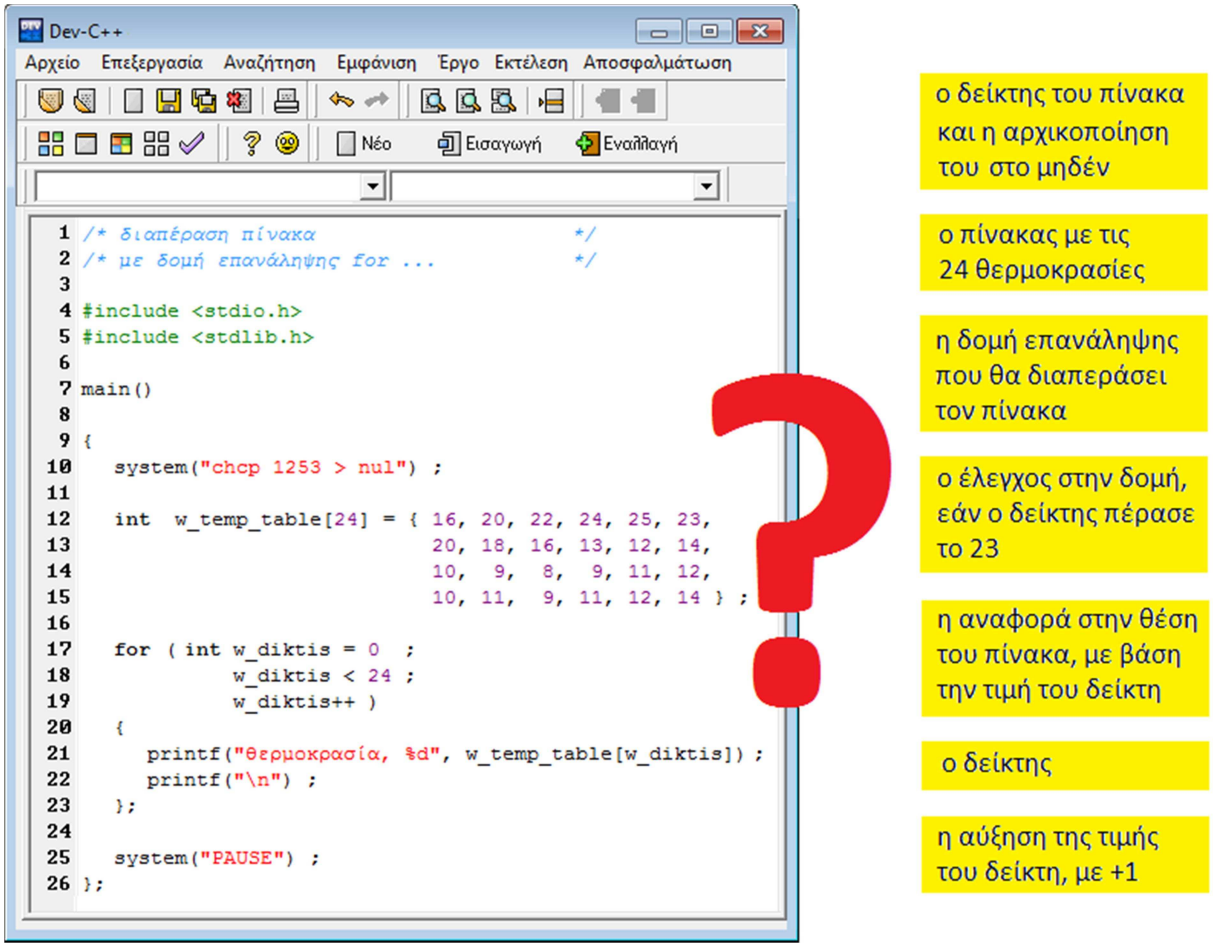


Ο υποβοηθητικός σχολιασμός για την προσέγγιση του πίνακα με την εντολή **do…while**




```
1  /* διαπέραση πίνακα                    */
2  /* με δομή επανάληψης for ...          */
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  main()
8
9  {
10     system("chcp 1253 > nul") ;
11
12     int  w_temp_table[24] = { 16, 20, 22, 24, 25, 23,
13                               20, 18, 16, 13, 12, 14,
14                               10,  9,  8,  9, 11, 12,
15                               10, 11,  9, 11, 12, 14 } ;
16
17     for  ( int w_diktis = 0  ;
18              w_diktis < 24 ;
19              w_diktis++ )
20     {
21         printf("θερμοκρασία, %d", w_temp_table[w_diktis]) ;
22         printf("\n") ;
23     };
24
25     system("PAUSE") ;
26 };
```

Η προσέγγιση του πίνακα με την εντολή **for…**



Τοποθετήστε κάθε μια από τις 7 κίτρινες ετικέτες σχολιασμού, στα κατάλληλα σημεία του κώδικα που κατά την γνώμη σας, τα υλοποιεί, η εντολή **for …**.

2.6. Διαπέραση Πινάκων

Η προσέγγιση ενός πίνακα για την αναζήτηση ενός συγκεκριμένου στοιχείου το οποίο μπορεί να βρίσκεται σε κάποια θέση του πίνακα, ονομάζεται *διαπέραση πίνακα* (search). Οι γλώσσες προγραμματισμού διαθέτουν διάφορες τεχνικές για την διαπέραση των πινάκων, με μερικές από αυτές τις τεχνικές να είναι,

- η γραμμική αναζήτηση σε διατεταγμένο πίνακα,
- η γραμμική αναζήτηση σε μη διατεταγμένο πίνακα,
- η δυαδική αναζήτηση σε διατεταγμένο πίνακα.

Στην γραμμική αναζήτηση σε διατεταγμένο πίνακα, καθώς τα στοιχεία του πίνακα είναι τοποθετημένα κατά αύξουσα ή και κατά φθίνουσα σειρά, μπορούμε να σταματήσουμε την

διαπέραση, όταν φτάσουμε σε στοιχείο που είναι μεγαλύτερο ή μικρότερο από αυτό που αναζητούμε. Στην γραμμική αναζήτηση σε μη διατεταγμένο πίνακα, θα πρέπει να διαπεράσουμε ολόκληρο τον πίνακα για να είμαστε σίγουροι ότι θα βρούμε ή όχι αυτό που αναζητούμε. Στην δυαδική αναζήτηση σε διατεταγμένο πίνακα, εφαρμόζουμε την τεχνική του «διαίρει και βασίλευε», ξεκινώντας από την μέση του πίνακα και εάν αυτό που αναζητούμε είναι μικρότερο από την τιμή που βρίσκουμε στην μέση του πίνακα, επαναλαμβάνουμε την διαδικασία στην αριστερή πλευρά του πίνακα, ενώ εάν είναι μεγαλύτερο από την τιμή στην μέση του πίνακα, επαναλαμβάνουμε την διαδικασία στην δεξιά πλευρά του πίνακα. Την τεχνική την συνεχίζουμε μέχρι να εξαντλήσουμε τα στοιχεία ή να βρούμε αυτό που αναζητάμε.

2.7. Αναδιάταξη Πινάκων

Η διαχείριση των πινάκων στις γλώσσες προγραμματισμού περιλαμβάνει και την έννοια της αναδιάταξης ή ταξινόμησης των στοιχείων τους. Η έννοια *αναδιάταξη*, σημαίνει την τακτοποίηση των στοιχείων του πίνακα με μια συγκεκριμένη σειρά. Η τακτοποίηση αυτή μπορεί να είναι αύξουσα ή φθίνουσα και μπορεί να αφορά αριθμητικά, αλφαβητικά ή αλφαριθμητικά δεδομένα. Οι αλγόριθμοι αναδιάταξης των στοιχείων στους πίνακες στις γλώσσες προγραμματισμού χρησιμοποιούνται ευρέως, με τους, bubble sort, selection sort και quick sort, να θεωρούνται οι περισσότερο γνωστοί.

2.8. Πίνακες πολλών Διαστάσεων

Οι πίνακες στις γλώσσες προγραμματισμού μπορούν να περιλαμβάνουν δομές οι οποίες αποτυπώνονται δομικά σε περισσότερες από μια διαστάσεις. Αυτό μπορεί να γίνει εύκολα αντιληπτό με έναν πίνακα δύο διαστάσεων, όπως ένα φύλο excel, με την πρώτη διάσταση να είναι, οι γραμμές και η δεύτερη διάσταση να είναι, οι στήλες του excel. Οι γλώσσες προγραμματισμού μπορούν και χειρίζονται πίνακες πολλών διαστάσεων. Οι πίνακες πολλών διαστάσεων χρησιμοποιούνται πολύ συχνά από τους έμπειρους προγραμματιστές.

Για παράδειγμα, η δήλωση,

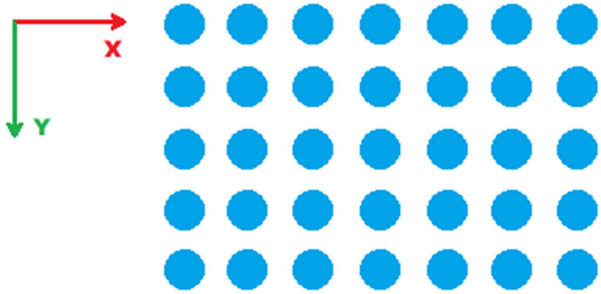
```
int  w_1d_my_pinakas[7] ;
```

δημιουργεί στην C++, έναν πίνακα, μιας διάστασης, με 7 διαθέσιμες θέσεις, για αποθήκευση ακέραιων αριθμών,

ενώ, η δήλωση,

```
int  w_2d_my_pinakas[5][7] ;
```

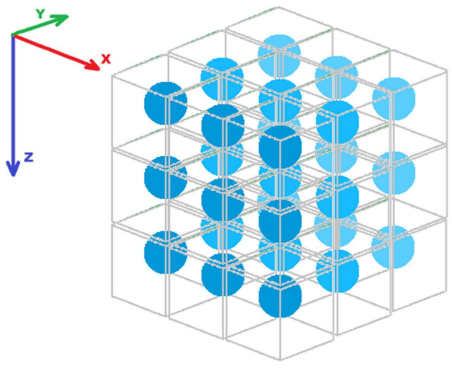
δημιουργεί στην C++, έναν πίνακα, δυο διαστάσεων, με 5 διαθέσιμες γραμμές και 7 διαθέσιμες στήλες ανά γραμμή, για την αποθήκευση ακέραιων αριθμών,



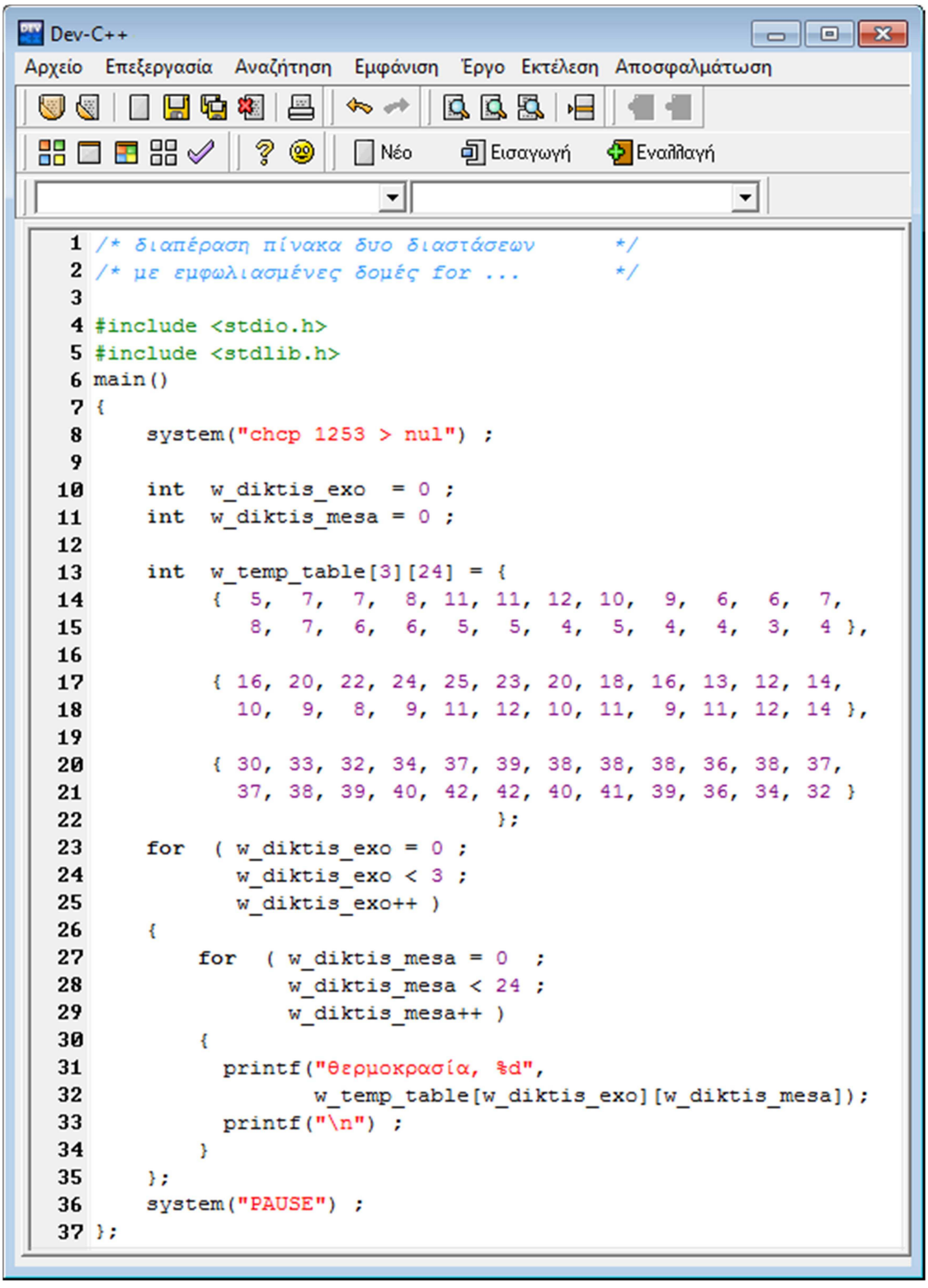
ενώ, η δήλωση,

```
int  w_3d_my_pinakas[3][3][3] ;
```

δημιουργεί στην C++, έναν πίνακα, τριών διαστάσεων, με 3 x 3 x 3, διαθέσιμες θέσεις, για την αποθήκευση ακέραιων αριθμών,



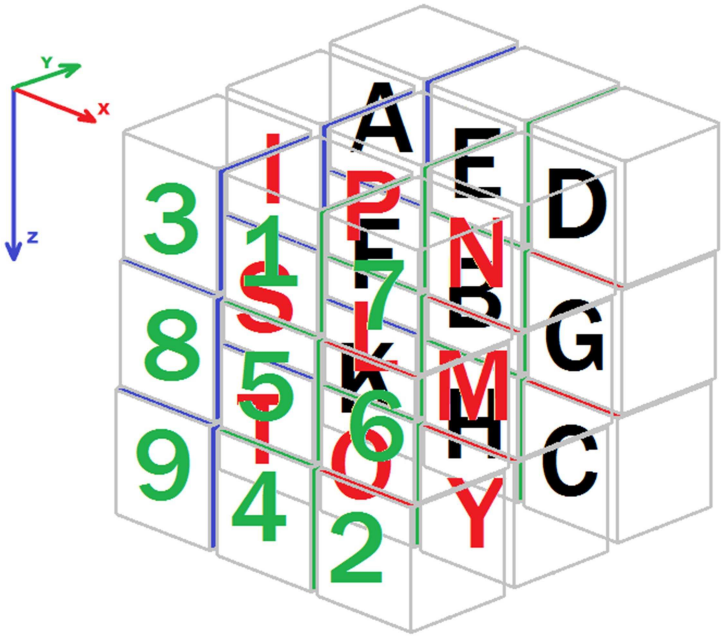
Την διαπέραση ενός πίνακα πολλών διαστάσεων μπορούμε να την υλοποιήσουμε στην C++, με εντολές δομών επανάληψης οι οποίες εμφωλιάζονται μέσα σε άλλες δομές επανάληψης.

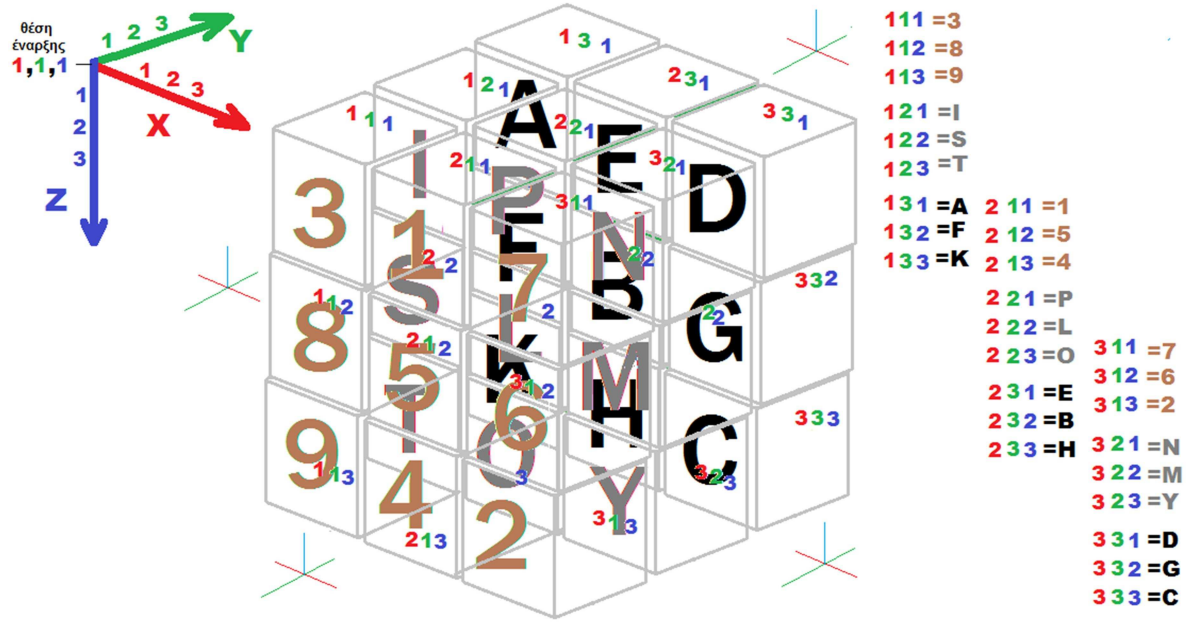



```
1  /* διαπέραση πίνακα δυο διαστάσεων       */
2  /* με εμφωλιασμένες δομές for ...        */
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  main()
7  {
8      system("chcp 1253 > nul") ;
9
10     int  w_diktis_exo  = 0 ;
11     int  w_diktis_mesa = 0 ;
12
13     int  w_temp_table[3][24] = {
14          {  5,  7,  7,  8, 11, 11, 12, 10,  9,  6,  6,  7,
15             8,  7,  6,  6,  5,  5,  4,  5,  4,  4,  3,  4 },
16
17          { 16, 20, 22, 24, 25, 23, 20, 18, 16, 13, 12, 14,
18            10,  9,  8,  9, 11, 12, 10, 11,  9, 11, 12, 14 },
19
20          { 30, 33, 32, 34, 37, 39, 38, 38, 38, 36, 38, 37,
21            37, 38, 39, 40, 42, 42, 40, 41, 39, 36, 34, 32 }
22                                  };
23     for  ( w_diktis_exo = 0 ;
24            w_diktis_exo < 3 ;
25            w_diktis_exo++ )
26     {
27         for  ( w_diktis_mesa = 0  ;
28                w_diktis_mesa < 24 ;
29                w_diktis_mesa++ )
30         {
31           printf("Θερμοκρασία, %d",
32                  w_temp_table[w_diktis_exo][w_diktis_mesa]);
33           printf("\n") ;
34         }
35     };
36     system("PAUSE") ;
37 };
```

Για παράδειγμα, σε έναν πίνακα 2 διαστάσεων ο οποίος φιλοξενεί τις 24 θερμοκρασίες από 3 ημέρες καταγραφής, θα χρειαστούμε δυο εμφωλιασμένες εντολές **for…**, με την εξωτερική να ανακυκλώνει τις γραμμές του πίνακα και την εσωτερική, να ανακυκλώνει τις στήλες του πίνακα. Ο κώδικας στην προηγούμενη εικόνα αναπαριστά ακριβώς αυτόν τον αλγόριθμό.

Έχοντας μελετήσει από το προηγούμενο παράδειγμα, τον κώδικα της δημιουργίας και της διαχείρισης του πίνακα των 2 διαστάσεων με τις 24 θερμοκρασίες και τις 3 ημέρες καταγραφής, θα μπορούσαμε να επεκτείνουμε τον διπλό εμφωλιασμό, σε τριπλό, με έναν πίνακα 3 διαστάσεων. Ειδικότερα, έχοντας στην διάθεση μας έναν πίνακα 3 διαστάσεων, τύπου **char**, με περιεχόμενο, όπως αυτό αποτυπώνεται στην επόμενη εικόνα, πως θα έπρεπε να είναι ένας κώδικας C++, που θα τύπωνε αυτά τα στοιχεία, α) από την αρχή του πίνακα προς το τέλος και β) από το τέλος του πίνακα προς την αρχή, θεωρώντας για αρχή του πίνακα, την θέση 1,1,1 και για τέλος την θέση 3,3,3. Στην πρώτη περίπτωση δηλαδή η διαπέραση του πίνακα να γίνει από την αρχή, και στην δεύτερη περίπτωση η διαπέραση να γίνει ανάποδα από το τέλος.





2.9. Πίνακες τύπου char και Αλφαριθμητικές Μεταβλητές

Η γλώσσα C++, διαθέτει κατάλληλες βιβλιοθήκες και πολλούς πρόσθετους τύπους δεδομένων για τον χειρισμό αλφαριθμητικών μεταβλητών. Με τον όρο *αλφαριθμητική μεταβλητή*, ορίζουμε μέσα σε ένα πρόγραμμα C++, την ύπαρξη μιας μεταβλητής (ενός πεδίου) το οποίο υποδέχεται σαν περιεχόμενο, μια συμβολοσειρά συγκεκριμένου μήκους. Για παράδειγμα, η μεταβλητή (το πεδίο), με το όνομα **w_eponimo**, θα μπορούσε να χρησιμοποιηθεί σε ένα πρόγραμμα C++, για να δημιουργηθεί μια διεπαφή η οποία θα ζητά από τον χρήστη, το επώνυμο του, το οποίο δεν θα πρέπει να ξεπερνά σε μήκος τους 50 χαρακτήρες. Αυτές οι αλφαριθμητικές μεταβλητές μπορούν να υλοποιηθούν και να διαχειριστούν στην C++, με πίνακες τύπου **char**. Ειδικά για τους πίνακες αλφαριθμητικού περιεχομένου **char**, η ανάθεση γίνεται με έναν διαφορετικό τρόπο, σε σχέση, με την ανάθεση σε αριθμητικούς πίνακες. Η γενική μορφή δήλωσης και ανάθεσης αρχικών τιμών σε έναν αλφαριθμητικό πίνακα στην γλώσσα C++, είναι,

char όνομα_πίνακα [θέσεις πίνακα] = "αλφαριθμητικό_περιεχόμενο" ;

Το αλφαριθμητικό περιεχόμενο βρίσκεται ανάμεσα σε εισαγωγικά " " και το πλήθος των χαρακτήρων θα πρέπει να είναι ίσο ή μικρότερο από τον αριθμό των θέσεων του πίνακα (το μέγεθος του πίνακα).

Για παράδειγμα, η δημιουργία και η ανάθεση με μια εντολή, ενός αλφαριθμητικού πίνακα ο οποίος θέλουμε να έχει για αλφαριθμητικό περιεχόμενο την έκφραση, ΕΛΛΑΣ-ΓΑΛΛΙΑ, μπορεί να γίνει με τις δηλώσεις,

(1) **char w_mystr_a[13] = "ΕΛΛΑΣ-ΓΑΛΛΙΑ";**

(2) **char w_mystr_b[13] = {'Ε', 'Λ', 'Λ', 'Α', 'Σ', '-', 'Γ', 'Α', 'Λ', 'Λ', 'Ι', 'Α', '\0'};**

(3) **char w_mystr_a[ ] = "ΕΛΛΑΣ-ΓΑΛΛΙΑ";**

Στην πρώτη δήλωση (1), ο πίνακας γεμίζει με μια ολική δήλωση περιεχομένου, σαν αλφαριθμητική σταθερά, μέσα σε διπλά εισαγωγικά **" "**. Ενώ στην δεύτερη δήλωση (2), ο πίνακας γεμίζει με λίστα τιμών (χαρακτήρα-χαρακτήρα) ανάμεσα σε άγκιστρα **{ }**, σαν μια παράθεση συμβατών τιμών μέσα σε μονά εισαγωγικά **' '**, οι οποίες τιμές διαχωρίζονται μεταξύ τους με κόμμα **,** και στο τέλος να τοποθετείται και η ακολουθία διαφυγής **\0**. Στην γλώσσα C++, το περιεχόμενο των πεδίων (μεταβλητών) τύπου **char**, πρέπει να τερματίζει με τον ειδικό χαρακτήρα διαφυγής (null), ο οποίος αποτυπώνεται με την ακολουθία διαφυγής **\0**. Στην τρίτη δήλωση (3), ο πίνακας γεμίζει πάλι με ολική δήλωση περιεχομένου, σαν αλφαριθμητική σταθερά, μέσα σε διπλά εισαγωγικά **" "**, και στην περίπτωση αυτή η δήλωση μεγέθους του πίνακα παραλείπεται διότι υπολογίζετε αυτόματα από την γλώσσα. Ειδικά για τις δηλώσεις (1) και (3), η γλώσσα έχει την ικανότητα και αυτόματα τοποθετεί και τον ειδικό χαρακτήρα διαφυγής (null), **\0**. Ποιο συγκεκριμένα, και για να γίνει αντιληπτή η διαφορά, στην σημασία των διπλών εισαγωγικών **" "**, από τα μονά εισαγωγικά **' '**. Στα διπλά εισαγωγικά η γλώσσα θεωρεί ότι «βλέπει» μια αλφαριθμητική σταθερά και εάν υποθέσουμε ότι είχαμε την δήλωση **"W"**, ο compiler θα δέσμευε ένα Byte, για να τοποθετήσει το γράμμα **W**, αλλά και ένα επιπλέον Byte, για να τοποθετήσει αυτόματα τον ειδικό χαρακτήρα διαφυγής (null), **\0**. Στην περίπτωση των μονών εισαγωγικών, η γλώσσα θεωρεί ότι «βλέπει» μια απλή σταθερά και εάν υποθέσουμε ότι είχαμε την δήλωση **'W'**, ο compiler θα δέσμευε ένα Byte, για να τοποθετήσει το γράμμα **W**, και δεν θα τοποθετούσε αυτόματα τον ειδικό χαρακτήρα διαφυγής (null), **\0**. Για παράδειγμα, μέσα σε ένα πεδίο (μεταβλητή) που θέλουμε να υποδέχεται ονόματα 12 χαρακτήρων, θα πρέπει να υπολογίσουμε και έναν επιπλέον χαρακτήρα (12+1=13) ο οποίος θα περιέχει την διαφυγή **\0**, και θα σηματοδοτεί το τέλος του αλφαριθμητικού περιεχομένου. Στην περίπτωση που αναθέσουμε στην μεταβλητή με ολική δήλωση περιεχομένου, σαν

αλφαριθμητική σταθερά, μέσα σε διπλά εισαγωγικά **“ “**, η γλώσσα τοποθετεί μόνη της και την διαφυγή **\0** . Αυτός είναι και ο λόγος που στην πρώτη δήλωση (1), **“ΕΛΛΑΣ-ΓΑΛΛΙΑ”**, η δήλωση δεσμεύει 13 θέσεις και όχι 12. Συνεπώς, εάν θέλαμε να βρούμε το πραγματικό τέλος στο μήκος μιας αλφαριθμητικής σταθεράς, ή αλλιώς ενός αλφαριθμητικού πίνακα string, θα έπρεπε να αναζητήσουμε το τελικό **\0**.

Σύνοψη κεφαλαίου

Στο δεύτερο κεφάλαιο του συγγράμματος, έγινε μια εκτενής αναφορά στην σημασία των οργανωμένων μορφών δεδομένων και αναπτύχθηκαν ειδικά θέματα για τους πίνακες και την γλώσσα προγραμματισμού C++. Οι εκπαιδευόμενοι καθοδηγήθηκαν με συστηματικό τρόπο στην ανάπτυξη δεξιοτήτων, σχετικών με την δημιουργία, χρήση και σημασία των πινάκων για την γλώσσα προγραμματισμού C++.

Ερωτήσεις αξιολόγησης

Ερώτηση-1: Περιγράψτε, τι σημαίνει ο όρος, *οργανωμένες δομές δεδομένων* και δώστε 3 παραδείγματα;

Ερώτηση-2: Περιγράψτε, τι σημαίνουν οι όροι, *γραμμική δομή δεδομένων* και *μη γραμμική δομή δεδομένων*;

Ερώτηση-3: Περιγράψτε, τι σημαίνουν οι όροι, *στατική δομή δεδομένων* και *δυναμική δομή δεδομένων*;

Ερώτηση-4: Ποιες είναι οι ενέργειες με τις οποίες μπορούμε να προσεγγίσουμε μια οργανωμένη δομή δεδομένων;

Ερώτηση-5: Δώστε την εντολή σε C++, με την οποία θα μπορούσατε να δηλώσετε έναν πίνακα 20 θέσεων, τύπου **float**;

Ερώτηση-6: Δώστε την εντολή σε C++, με την οποία θα μπορούσατε να προσεγγίσετε την πρώτη θέση του πίνακα που δημιουργήσατε με την ερώτηση_5;

Ερώτηση-7: Δώστε την εντολή σε C++, με την οποία θα μπορούσατε να προσεγγίσετε την τελευταία θέση του πίνακα που δημιουργήσατε με την ερώτηση_5;

Ερώτηση-8: Δώστε την εντολή σε C++, με την οποία θα μπορούσατε να δηλώσετε έναν δυσδιάστατο πίνακα 21x11 θέσεων, τύπου **int**;

Ερώτηση-9: Είναι σωστό, ότι πρώτα δηλώνουμε τον πίνακα και αμέσως μετά μπορούμε να δηλώσουμε τον δείκτη που θα χρησιμοποιούμε για την διαπέραση του;

Ερώτηση-10: Εάν δηλώσουμε ένα πίνακα με την εντολή **int w_my_table[11]** και τον προσεγγίσουμε με την εντολή **w_temp_table[11]**, τι θα συμβεί και γιατί;

Ερώτηση-11: Δώστε την εντολή σε C++, με την οποία θα μπορούσατε να προσεγγίσετε την πρώτη θέση ενός τρισδιάστατου πίνακα 30x20x10 θέσεων, τύπου **int**, με το όνομα **w_3d_table**;

Ερώτηση-12: Είναι σωστή, η εντολή, **int w_my_new_str_b[5] = {'Ε', 'Λ', 'Λ', 'Α', 'Σ'};**;

Ερώτηση-13: Με ποια αλγοριθμική δομή θα μπορούσατε να διαπεράσετε έναν τρισδιάστατο πίνακα;

Ερώτηση-14: Στην γλώσσα προγραμματισμού C++, μπορούμε να δημιουργήσουμε πίνακες πέντε διαστάσεων;

Απαντήσεις ερωτήσεων αξιολόγησης

Απάντηση-1: Συμβουλευτείτε τις εισαγωγικές παρατηρήσεις και την ενότητα 2.3. του κεφαλαίου.

Απάντηση-2: Συμβουλευτείτε την ενότητα 2.1. του κεφαλαίου.

Απάντηση-3: Συμβουλευτείτε την ενότητα 2.1. του κεφαλαίου.

Απάντηση-4: Συμβουλευτείτε την ενότητα 2.2. του κεφαλαίου.

Απάντηση-5: **float w_my_table[20]**.

Απάντηση-6: **w_my_table[0]**.

Απάντηση-7: **w_my_table[19]**.

Απάντηση-8: **int w_my_table[21][11]**.

Απάντηση-9: Ναι, είναι σωστό, αρκεί την διαπέραση να την βάλουμε μετά από την δήλωση του πίνακα και του δείκτη.

Απάντηση-10: Το πρόγραμμα θα «σκάσει», καθώς θα βγει εκτός των ορίων του πίνακα, διότι ο αριθμός 11, δηλώνει την θέση 12, η οποία δεν υπάρχει, αφού ο πίνακας έχει μόνο 11 θέσεις.

Απάντηση-11: **w_3d_table[0][0][0]**.

Απάντηση-12: Όχι , υπάρχουν δυο λάθη, το πρώτο είναι η ασυμβατότητα του τύπου **int**, με το περιεχόμενο που είναι χαρακτήρες και το δεύτερο είναι, το μέγεθος 5, που δεν επιτρέπει πρόσθετη θέση για τον ειδικό χαρακτήρα διαφυγής (null), **\0**, στο τέλος της δήλωσης δημιουργίας του πίνακα.

Απάντηση-13: Και με τις τρεις αλγοριθμικές δομές επανάληψης και εμφωλιάζοντας την μια μέσα στην άλλη, για 3 επίπεδα.

Απάντηση-14: Ναι μπορούμε, για παράδειγμα η δήλωση, **int w_5d_table[5] [10] [10] [5] [5]**, δημιουργεί έναν πίνακα πέντε διαστάσεων, 5x10x10x5x5 θέσεων, τύπου **int**, με το όνομα **w_5d_table**.

3. ΣΥΝΑΡΤΗΣΕΙΣ

Σκοπός και επιμέρους στόχοι

Ο σκοπός του σημαντικού αυτού κεφαλαίου είναι να μεταφέρει στους εκπαιδευόμενους την απαραίτητη τεχνική γνώση για να διασπούν ένα πρόγραμμα σε ανεξάρτητα αυτόνομα τμήματα κώδικα C++, επιμερίζοντας σε κομμάτια, την πολυπλοκότητα ενός σύνθετου αλγορίθμου. Να μπορούν στην συνέχεια, να διασύνδεουν καταλλήλως αυτές τις μονάδες, μεταβιβάζοντας το περιεχόμενο των μεταβλητών με ενδεδειγμένους τρόπους.

Προσδοκώμενα αποτελέσματα

Με την μελέτη του τρίτου κεφαλαίου οι εκπαιδευόμενοι θα μπορούν,

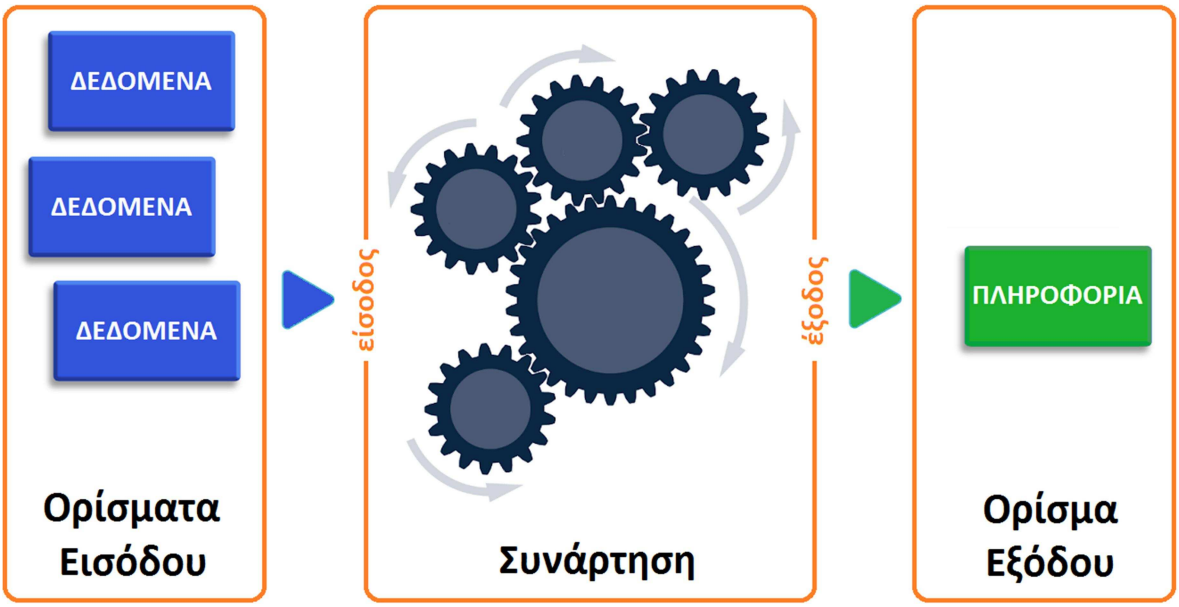
- να διασπούν τα προγράμματα τους σε μικρότερες αυτόνομες μονάδες κώδικα,
- να συντάσσουν μια συνάρτηση που θα εκτελεί μια συγκεκριμένη αυτόνομη λειτουργία,
- να τοποθετούν το σώμα μιας συνάρτησης στο κατάλληλο σημείο ενός προγράμματος C++,
- να περιγράφουν τι είναι τα ορίσματα εισόδου και τι τα ορίσματα εξόδου, των συναρτήσεων, στην γλώσσα προγραμματισμού C++,
- να συντάσσουν μια συνάρτηση που δεν θα έχει ορίσματα εξόδου,
- να τοποθετούν πολλές συναρτήσεις μέσα σε ένα πρόγραμμα C++,
- να χειρίζονται τις επιστροφές μιας συνάρτησης,
- να καταλαβαίνουν πως μπορούν να μεταβιβάζουν παραμέτρους μεταξύ των τμημάτων του κώδικα τους,
- να περιγράφουν την έννοια της εμβέλειας των μεταβλητών στην γλώσσα C++,
- να αντιλαμβάνονται την ορατότητα και την διάρκεια ζωής των μεταβλητών στην C++,
- να περιγράψουν τι σημαίνει τοπική μεταβλητή και τι σημαίνει καθολική μεταβλητή, μέσα σε ένα πρόγραμμα C++.

Έννοιες – Λέξεις Κλειδιά

Συναρτήσεις, Ορίσματα εισόδου, Ορίσματα εξόδου, Επιστρεφόμενη μεταβλητή, Εμβέλεια μεταβλητής, Επιστροφές συναρτήσεων, Μεταβλητή τοπικής εμβέλειας, Μεταβλητή καθολικής εμβέλειας, Διάρκεια ζωής μεταβλητής, Μερική διάρκεια ζωής μεταβλητής, Πλήρης διάρκεια ζωής μεταβλητής.

Εισαγωγικές Παρατηρήσεις

Στο 2^{ο} σύγγραμμα της σειράς, *Προγραμματισμός Ηλεκτρονικών Υπολογιστών & Μηχανών*, της ACTA (κεφάλαιο 2, ενότητα 7), έγινε μια εισαγωγική παρουσίαση του θέματος της διάσπασης των μεγάλων προγραμμάτων σε επιμέρους τμήματα με στόχο την απλοποίηση της διαδικασίας ανάπτυξης και συντήρησης ενός συστήματος λογισμικού. Αυτή η πρακτική της οργάνωσης του κώδικα σε τμήματα, με βάση την λογική του «*διαίρει και βασίλευε*», θεωρείται απολύτως αναγκαία τακτική, για το *διαδικασιακό* μοντέλο και φυσικά την γλώσσα προγραμματισμού C++. Στο νέο αυτό κεφάλαιο, το θέμα της διάσπασης ενός προγράμματος σε επιμέρους λειτουργικά κομμάτια, επεκτείνεται περαιτέρω και αναλύεται διαμέσου των δυνατοτήτων της γλώσσας προγραμματισμού C++.



Η οργάνωση ενός προγράμματός C++, σε τμήματα, που ονομάζονται *συναρτήσεις* (functions), είναι βασικό βήμα για την επίλυση ενός αλγορίθμου. Οι μικρότερες αυτές αυτόνομες μονάδες κώδικα μπορούν να υλοποιούν συναρτήσεις, με συγκεκριμένο όνομα, συγκεκριμένα δεδομένα εισόδου (ορίσματα εισόδου) και να έχουν σκοπό να επιτελούν συγκεκριμένες εργασίες, παράγοντας συγκεκριμένα αποτελέσματα (ορίσματα εξόδου). Οι συναρτήσεις στον προγραμματισμό, ονομάζονται επίσης, υποπρογράμματα, subprograms, ρουτίνες, υπορουτίνες, μέθοδοι και methods. Στην γλώσσα C++, ένα πρόγραμμα ξεκινά πάντα από την πρώτη βασική συνάρτηση με το όνομα, **main**.

3.1. Δημιουργώντας μια Συνάρτηση

Η γενική μορφή σύνταξης για την δημιουργία μιας συνάρτησης στην γλώσσα C++, είναι,

```
τύπος_επιστρεφόμενης_μεταβλητής  όνομα_συνάρτησης
                                 (
                                 τύπος_ορίσματος_A όνομα_ορίσματος_A,
                                 τύπος_ορίσματος_B όνομα_ορίσματος_B,
                                 …
                                 τύπος_ορίσματος_N όνομα_ορίσματος_N
                                 )
{
 … σώμα εντολών που θα εκτελεστούν όταν κληθεί η συνάρτηση
};
```

Για παράδειγμα, η επόμενη δομή εντολών είναι μια ολοκληρωμένη συνάρτηση,

```
int  my_routina
                                ( int w_metritis_x,
            int w_metritis_y,
            int w_metritis_z )
{
  int w_synolo = 0 ;
  printf( "PEDIO_x=%d    PEDIO_y=%d   PEDIO_z=%d",
              w_metritis_x,  w_metritis_y,  w_metritis_z );
  w_synolo = w_metritis_x + w_metritis_y + w_metritis_z ;
  return w_synolo ;
};
```

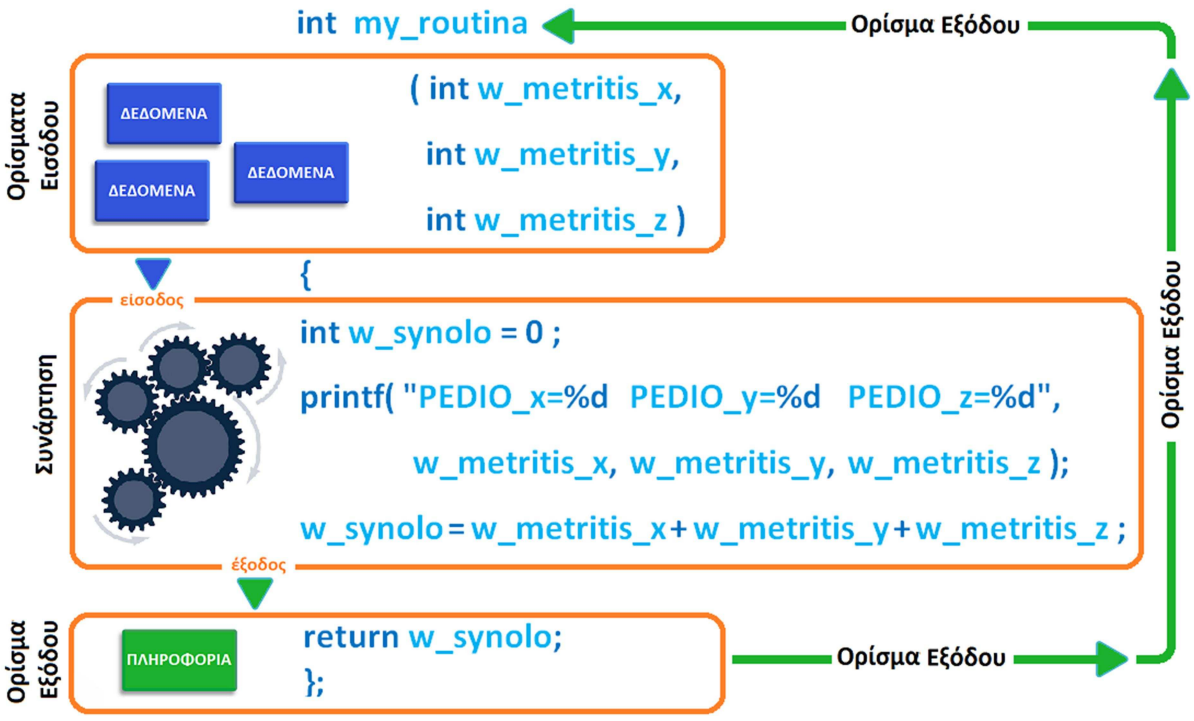
- με το όνομα, **my_routina**,
- τα ορίσματα εισόδου, **w_diktis_x, w_diktis_y, w_diktis_z**, όλα τύπου **int**,

- που εκτελεί μια εντολή εκτύπωσης **printf**, των 3 ορισμάτων εισόδου και συνεχίζει με μια άθροιση των 3 ορισμάτων εισόδου, στην μεταβλητή **w_synolo**, που ορίστηκε τοπικά μέσα στην ίδια την συνάρτηση,
- και ένα όρισμα εξόδου, τύπου **int**, στο οποίο «στέλνεται» στην έξοδο της συνάρτησης, η τιμή από την άθροιση των 3 ορισμάτων εισόδου, με την εντολή **return w_synolo**.

3.2. Πως λειτουργεί μια Συνάρτηση

Η κλήση μιας συνάρτησης γίνεται με το όνομά της, **my_routina(5, 10, 20)** και μέσα σε παρενθέσεις **( )**, τις κατάλληλες μεταβλητές ή τις τιμές κατάλληλου τύπου (5, 10, 20) με τα δηλωμένα ορίσματα εισόδου της συνάρτησης. Το πλήθος των στοιχείων μέσα στις παρενθέσεις πρέπει να ταιριάζει σε πλήθος αλλά και στους τύπους με αυτά που ορίστηκαν στην δήλωση της συνάρτησης.

Σε συνέχεια της ενότητας 3.1., η συνάρτηση **my_routina**, όταν κληθεί, εκτελεί μια εκτύπωση και μια άθροιση, στέλνοντας στην μεταβλητή **w_synolo**, σαν όρισμα εξόδου το αποτέλεσμα της πράξης της άθροισης.

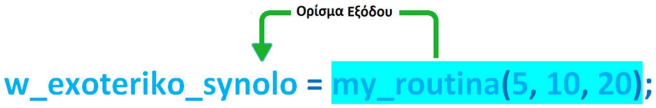


Η μεταβλητή **w_synolo**, είναι μια μεταβλητή περιορισμένης τοπικής εμβέλειας, η οποία δημιουργήθηκε ειδικά μέσα στην συνάρτηση και για το λόγο αυτό, δεν είναι ορατή από το υπόλοιπο πρόγραμμα. Όμως, το περιεχόμενο της, που επιστρέφεται σαν όρισμα εξόδου, μπορεί να χρησιμοποιηθεί, όπως θα δούμε ακολούθως. Την έννοια της εμβέλειας των μεταβλητών στην γλώσσα C++, θα μελετήσουμε αναλυτικά και στην συνέχεια.

Ένα λεπτό σημείο στην όλη διαδικασία που εκτελεί η γλώσσα C++ (και όλες οι ομοιάζουσες με αυτήν γλώσσες), αποτυπώνεται σχηματικά στην ανωτέρω εικόνα. Συγκεκριμένα, το «Όρισμα Εξόδου», το οποίο ενώ νοητικά βρίσκεται στο τέλος της συνάρτησης, στην πραγματικότητα η γλώσσα, με την ολοκλήρωση της λειτουργίας της συνάρτησης το ανακατευθύνει (το περιεχόμενο της **w_synolo**), προς την είσοδο της συνάρτησης, το οποίο αποτυπώνεται στην εικόνα με ένα συνεχόμενο πράσινο βέλος, με τον τίτλο «Όρισμα Εξόδου». Η γλώσσα με αυτή την «ανάποδη» κίνηση έχει στόχο να «επιστρέψει», το περιεχόμενο της **w_synolo** στο σημείο, του κύριου προγράμματος, από το οποίο έγινε η κλήση της συνάρτησης **my_routina**. Αυτό σημαίνει ότι στην συνέχεια, είναι αποκλειστική ευθύνη του προγραμματιστή να «παραλάβει», αυτό το περιεχόμενο της «επιστροφής», με μια εντολή ανάθεσης σε μια άλλη μεταβλητή, συμβατού τύπου **int**, με την είσοδο της συνάρτησης.

Την «παραλαβή» αυτή, υλοποιούν οι τρεις επόμενες εντολές. Η πρώτη εντολή, δημιουργεί την μεταβλητή **w_exoteriko_synolo** για την «παραλαβή» και η δεύτερη εντολή είναι μια ενέργεια ανάθεσης (το σύμβολο **=**), με την μεταβλητή στην αριστερή πλευρά και στην δεξιά πλευρά την συνάρτηση **my_routina** και ότι αυτή έχει παράγει από την λειτουργία της. Η δήλωση δεξιά από το σύμβολο της ισότητας, **my_routina(5, 10, 20)**, είναι το σημείο που γίνεται κλήση της συνάρτησης.

```
int  w_exoteriko_synolo = 0 ;
```

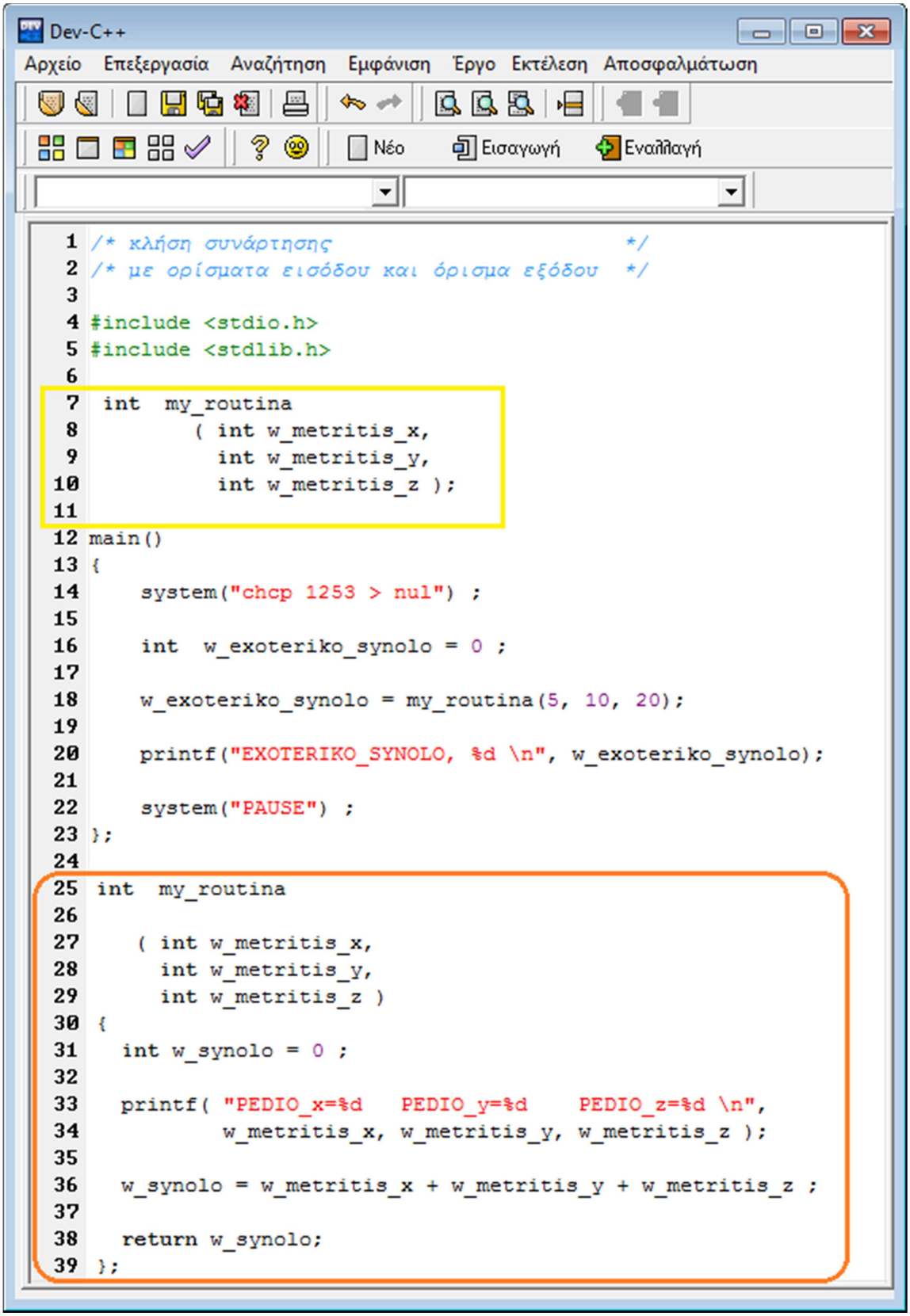



```
if  w_exoteriko_synolo …
```

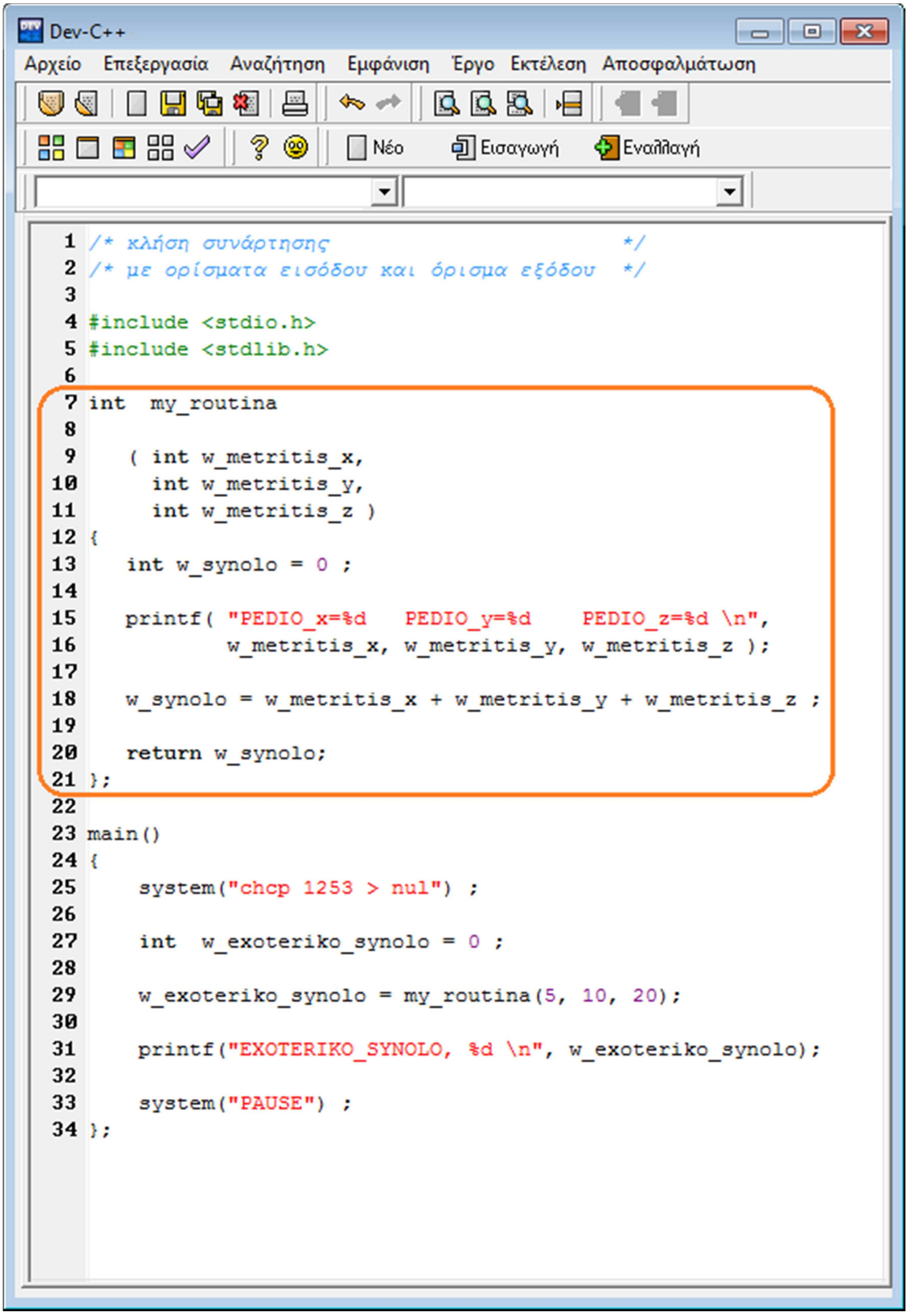
Με την επιστροφή της «μπίλιας» (της ροής εκτέλεσης), η μεταβλητή **w_exoteriko_synolo**, έχει «παραλάβει» το περιεχόμενο που έστειλε η εντολή **return w_synolo**, μέσα από την συνάρτηση. Με την αμέσως επόμενη εντολή, το **if**, ο κώδικας συνεχίζει με έλεγχο του περιεχομένου της μεταβλητής **w_exoteriko_synolo** και την εκτέλεση κάποιου κατάλληλου κώδικα.

Οι επιστροφές των συναρτήσεων της γλώσσας C++, δεν αποτελούν κάποια ιδιαίτερα δύσκολη κατάσταση και μπορούν εύκολα να αφομοιωθούν από τους εκπαιδευόμενους με την μελέτη και την παρατήρηση των δυο προηγούμενων εικόνων της ενότητας 3.7.

Ακολουθεί παράδειγμα σε κώδικα C++, το οποίο υλοποιεί πλήρως την λειτουργία συγγραφής, κλήσης και χρήσης του ορίσματος εξόδου μιας συνάρτησης. Επιπλέον, ο κώδικας αποτυπώνεται με δυο διαφορετικούς τόπους. Συγκεκριμένα, στην πρώτη περίπτωση ο κώδικας με ολόκληρο το σώμα των εντολών της **my_routina**, βρίσκεται μετά από την κεντρική ρουτίνα έναρξης **main**. Στην δεύτερη περίπτωση ο κώδικας με ολόκληρο το σώμα των εντολών της **my_routina**, βρίσκεται πριν από την κεντρική ρουτίνα έναρξης **main** και επιπλέον συνοδεύεται με μια επανάληψη της επικεφαλίδας της συνάρτησης επάνω και έξω από την **main** (σημειώνεται με το κίτρινο περιθώριο). Ο λόγος που ο κώδικας παρουσιάζεται με αυτούς τους τρόπους, είναι διότι στον συγκεκριμένο compiler Dev-C++, η γλώσσα πρέπει να γνωρίζει πριν αρχίσει να εκτελεί την **main**, ποιες είναι οι συναρτήσεις που θα κληθούν και θα χρησιμοποιηθούν στην συνέχεια. Συνεπώς, θα πρέπει με κάποιο τρόπο να τις γνωστοποιήσουμε στον compiler πριν από την **main**. Αυτό ακριβώς κάνουν οι δυο διαφορετικοί τρόποι γραφής του κώδικα, καθορίζουν ποιες είναι οι συναρτήσεις πριν καν αυτές κληθούν από την κεντρική. Το αποτέλεσμα και στις δυο περιπτώσεις θα είναι το ακριβώς το ίδιο. Αυτή η ιδιομορφία του συγκεκριμένου compiler δεν αφορά όλες τις ομοιάζουσες γλώσσες και όλους τους compiler.



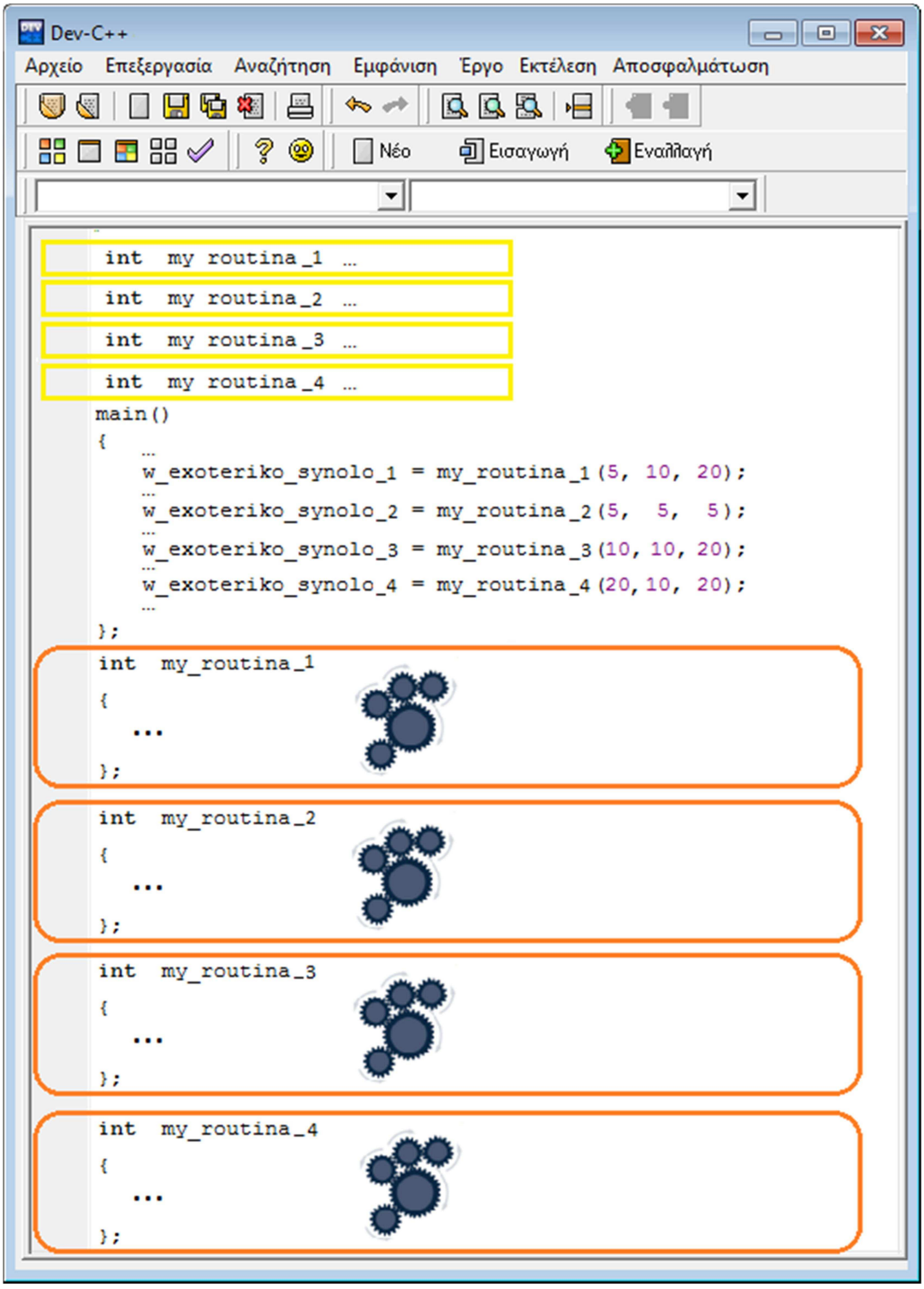
Κώδικας κλήσης συνάρτησης με προκαθορισμό, πριν από την **main**.



Κώδικας κλήσης συνάρτησης χωρίς προκαθορισμό, πριν από την **main**.

3.3. Πολλές Συναρτήσεις στην σειρά

Είναι προφανές ότι ένα πρόγραμμα μπορεί να περιέχει πολλές συναρτήσεις και στην περίπτωση αυτή, οι κανόνες συγγραφής αφορούν απλά και μόνο την τοποθέτηση της κάθε συνάρτησης κάτω από τις άλλες.



3.4. Συναρτήσεις χωρίς Επιστροφή

Μια συνάρτηση μπορεί να έχει ή και να μην έχει έξοδο. Στην περίπτωση που δεν θέλουμε να υπάρχει έξοδος, (κανένα όρισμα εξόδου) αντικαθιστούμε την αρχική δήλωση **τύπος_επιστρεφόμενης_μεταβλητής**, με την δεσμευμένη λέξη **void**. Στην περίπτωση αυτή, ο ίδιος προηγούμενος κώδικας θα έχει την επόμενη μορφή και δεν θα περιέχει επιστροφή.

```
void  my_routina
                              ( int w_metritis_x,
          int w_metritis_y,
          int w_metritis_z )
{
  int w_synolo = 0 ;
  printf( "PEDIO_x=%d    PEDIO_y=%d   PEDIO_z=%d",
           w_metritis_x,  w_metritis_y,  w_metritis_z );
  w_synolo = w_metritis_x + w_metritis_y + w_metritis_z ;
  printf( "SYNOLO=%d", w_synolo);
};
```

Συνεπώς η συνάρτηση δεν θα έχει επιστροφή με ανάθεση σε μεταβλητή και η κλήση της θα γίνεται απλά με την εντολή,

```
my_routina(5, 10, 20);
```

Μια συνάρτηση που δεν έχει επιστροφή και ορίζεται με **void**, μπορεί, χωρίς αυτό να είναι απαραίτητο, να έχει εντολές **return**, οι οποίες να μην περιέχουν καμία τιμή επιστροφής,

```
return ;
```

Επιπλέον, μια επιστροφή **return**, μπορεί να περιέχει μια ολόκληρη έκφραση, **return(w_poso_a * w_poso_b / 2)**, ή ακόμα και την κλήση μιας άλλης εμφωλιασμένης συνάρτησης.

3.5. Συναρτήσεις με πολλές Επιστροφές

Μια συνάρτηση μπορεί να έχει περισσότερες από μια εξόδους. Στην περίπτωση αυτή μέσα στο σώμα των εντολών της συνάρτησης τοποθετούμε σε διάφορα κατάλληλα σημεία την εντολή **return**, με την αντίστοιχη τιμή της. Σε κάποια από αυτά τα **return**, μπορούμε να επιστρέφουμε μια μεταβλητή και σε κάποια άλλα μπορούμε να επιστρέφουμε μια άλλη μεταβλητή ή απλά μια τιμή.

```
int my_many_exits_routina  ( int w_poso_a, int w_poso_b )
{
   int w_synolo = 0 ;
   w_synolo = w_poso_a + w_poso_b ;

   if  ( w_synolo  > 1000 )
         { return w_synolo ; }
   else
         { return 500 ; }

};
```

3.6. Συναρτήσεις και Εμβέλεια Μεταβλητών

Στην γλώσσα προγραμματισμού C++, η διάσπαση ενός προγράμματος σε επιμέρους τμήματα κώδικα δημιουργεί το θέμα της εμβέλειας των μεταβλητών. Οι μεταβλητές με βάση την εμβέλεια που αναλαμβάνουν διαχωρίζονται σε,

- τοπικές μεταβλητές (local variables)
- καθολικές μεταβλητές (global variables)

Μια μεταβλητή με το όνομα **w_var_a**, η οποία ορίζεται μέσα σε μια συνάρτηση **synart_01**, θεωρείται τοπική μεταβλητή και είναι «ορατή» μόνο μέσα σε αυτή την συνάρτηση και μόνο κατά την διάρκεια της λειτουργίας της. Εάν μέσα σε μια άλλη συνάρτηση με το όνομα

synart_02, δηλώσουμε μια άλλη τοπική μεταβλητή, επίσης με το όνομα **w_var_a**, τα δυο αυτά αντικείμενα παρόλο που έχουν το ίδιο όνομα θα θεωρούνται διαφορετικές μεταβλητές και θα είναι η κάθε μια αναγνωρίσιμη μόνο για την συνάρτηση που ορίστηκαν.

Η μεταβλητή δηλαδή της πρώτης συνάρτησης είναι εντελώς άγνωστη για την δεύτερη συνάρτηση και η μεταβλητή της δεύτερης συνάρτησης είναι εντελώς άγνωστη για την πρώτη συνάρτηση, παρόλο που έχουν τα ίδια ονόματα. Και αυτό συμβαίνει διότι κατά την δημιουργία τους και την σύνδεση τους με την μνήμη του υπολογιστή, δεσμεύουν εντελώς διαφορετικές και ανεξάρτητες περιοχές μνήμης. Καθώς όλα αυτά διαδραματίζονται μέσω μιας κεντρικής συνάρτησης **main**, η «μπίλια» που θα βγει από την συνάρτηση, και θα επιστραφεί στην **main**, θα αφήσει κατεστραμμένο το περιεχόμενο της τοπικής μεταβλητής και για τον λόγο αυτό, εάν το περιεχόμενο είναι σημαντικό, θα πρέπει ο προγραμματιστής να φροντίζει να την έχει αναθέσει επάνω σε μια άλλη εξωτερική μεταβλητή στην **main**.
Οι τοπικές καθολικές μεταβλητές (local variables) θεωρείται ότι έχουν μερική διάρκεια ζωής, καθόσον τρέχει η συνάρτηση τους. Η έννοια της τοπικότητας αφορά και την κεντρική συνάρτηση **main**, όπου ισχύουν τα ίδια για τις μεταβλητές που φιλοξενεί. Στην περίπτωση που μια μεταβλητή δηλωθεί εκτός της κεντρικής συνάρτησης **main** και έξω από κάθε άλλη συνάρτηση, τότε αυτή θεωρείται καθολική μεταβλητή (global variable) και είναι αναγνωρίσιμη από κάθε σημείο του προγράμματος. Οι καθολικές μεταβλητές καταστρέφονται μόνο με το τέλος ολόκληρου του προγράμματος και θεωρείται ότι έχουν πλήρη διάρκεια ζωής. Ειδικά για την διάρκεια ζωής του περιεχομένου μιας μεταβλητής η C++, διαθέτει την πρόσθετη δήλωση **static** η οποία εάν τοποθετηθεί μπροστά από την δήλωση της μεταβλητής μέσα σε μια συνάρτηση, τότε επεκτείνει την διάρκεια ζωής της μεταβλητής μέχρι το τέλος ολόκληρου του προγράμματος

```
static  int w_stable_synolo;
```

Το περιεχόμενο της δηλαδή δεν καταστρέφεται με την έξοδο από την συνάρτηση αλλά παραμένει, με την προϋπόθεση βέβαια ότι δεν αρχικοποιείται στην επόμενη επανάκληση της συνάρτησης που την περιέχει.

Σύνοψη κεφαλαίου

Στο τρίτο κεφάλαιο αναπτύξαμε τις προχωρημένες έννοιες των συναρτήσεων και περιγράψαμε την ιδιαίτερη σημασία τους στην ανάπτυξη συστημάτων λογισμικού. Ειδικότερα μελετήσαμε σημαντικά θέματα των συναρτήσεων, σχετικά, με την εμβέλεια των μεταβλητών τους, τα επιλεγόμενα ονόματα και την διάρκεια της ζωής των μεταβλητών τους. Το κεφάλαιο συνόδευσαν ολοκληρωμένα λειτουργικά παραδείγματα κώδικα και επεξηγήσεις για πολλά ειδικά θέματα.

Ερωτήσεις αξιολόγησης

Ερώτηση-1: Περιγράψτε, τι σημαίνει ο όρος, *συνάρτηση* και για ποιον λόγο χρησιμοποιούμε τις συναρτήσεις;

Ερώτηση-2: Δώστε την έννοια της συνάρτησης, με άλλους 2-3 παρόμοιους όρους;

Ερώτηση-3: Περιγράψτε, τι είναι τα ορίσματα εισόδου;

Ερώτηση-4: Περιγράψτε, τι είναι τα ορίσματα εξόδου;

Ερώτηση-5: Περιγράψτε, με ποιον τρόπο μπορούμε να πάρουμε την επιστροφή μιας συνάρτησης;

Ερώτηση-6: Περιγράψτε, τι είναι καθολική μεταβλητή, μέσα σε ένα πρόγραμμα C++;

Ερώτηση-7: Περιγράψτε, τι είναι τοπική μεταβλητή, μέσα σε ένα πρόγραμμα C++;

Ερώτηση-8: Περιγράψτε, πως μπορείτε να επηρεάσετε την διάρκεια ζωής μιας τοπικής μεταβλητής;

Ερώτηση-9: Είναι σωστό ότι μια συνάρτηση πρέπει να έχει πάντα ορίσματα εισόδου;

Ερώτηση-10: Είναι σωστό ότι μια συνάρτηση πρέπει να έχει πάντα ορίσματα εξόδου;

Ερώτηση-11: Είναι σωστό, ότι μια συνάρτηση μπορεί να μην έχει καμία δήλωση επιστροφής **return**;

Ερώτηση-12: Είναι σωστό, ότι μπορούμε να καλέσουμε μια συνάρτηση με διαφορετικό αριθμό ορισμάτων από αυτόν που έχει δηλωθεί στον ορισμό της συνάρτησης;

Ερώτηση-13: Είναι σωστό, ότι μπορούμε να καλέσουμε μια συνάρτηση με διαφορετικό τύπο ορισμάτων από αυτόν που έχει δηλωθεί στον ορισμό της συνάρτησης;

Ερώτηση-14: Είναι σωστό, ότι μπορούμε να διατηρήσουμε για μια μεταβλητή που ορίστηκε μέσα σε μια συνάρτηση να έχει το ίδιο όνομα με μια άλλη μεταβλητή η οποία θα βρίσκεται στην κεντρική συνάρτηση **main**, ενός προγράμματος C++;

Απαντήσεις ερωτήσεων αξιολόγησης

Απάντηση-1: Συμβουλευτείτε τις εισαγωγικές παρατηρήσεις του κεφαλαίου.

Απάντηση-2: Συμβουλευτείτε τις εισαγωγικές παρατηρήσεις του κεφαλαίου.

Απάντηση-3: Συμβουλευτείτε τις εισαγωγικές παρατηρήσεις του κεφαλαίου.

Απάντηση-4: Συμβουλευτείτε τις εισαγωγικές παρατηρήσεις του κεφαλαίου.

Απάντηση-5: Στο σημείο που κάνουμε κλήση στην συνάρτηση μπορούμε να προσθέσουμε και μια ανάθεση σε μια κατάλληλη μεταβλητή. Συμβουλευτείτε την ενότητα 3.2. του κεφαλαίου.

Απάντηση-6: Συμβουλευτείτε την ενότητα 3.6. του κεφαλαίου.

Απάντηση-7: Συμβουλευτείτε την ενότητα 3.6. του κεφαλαίου.

Απάντηση-8: Συμβουλευτείτε την ενότητα 3.6. του κεφαλαίου.

Απάντηση-9: Όχι είναι λάθος, μια συνάρτηση μπορεί να μην έχει ορίσματα εισόδου.

Απάντηση-10: Όχι είναι λάθος, μια συνάρτηση μπορεί να μην έχει ορίσματα εξόδου.

Απάντηση-11: Ναι, είναι σωστό, μπορεί να μην έχει καμία δήλωση επιστροφής **return**.

Απάντηση-12: Όχι είναι λάθος, πρέπει η κλήση να έχει ίδιο αριθμό ορισμάτων με αυτόν που έχει δηλωθεί η συνάρτηση.

Απάντηση-13: Όχι είναι λάθος, πρέπει η κλήση να έχει ίδιο τύπο ορισμάτων με αυτόν που έχει δηλωθεί η συνάρτηση.

Απάντηση-14: Ναι, είναι σωστό.

4. ΕΙΔΙΚΑ ΘΕΜΑΤΑ ΤΗΣ C++

Σκοπός και επιμέρους στόχοι

Στο τέταρτο κεφάλαιο οι εκπαιδευόμενοι εισάγονται στο πεδίο των ειδικών θεμάτων της γλώσσας προγραμματισμού C++. Ειδικότερα, παρουσιάζονται με αναλυτικά επεξηγηματικά παραδείγματα, θέματα όπως, οι αναφορές, οι δείκτες μνήμης της C++ και παροτρύνονται οι αναγνώστες να συνεχίσουν την μελέτης επάνω στα ειδικά θέματα της γλώσσας.

Προσδοκώμενα αποτελέσματα

Με την μελέτη του τέταρτου κεφαλαίου οι εκπαιδευόμενοι θα μπορούν,

- να δημιουργούν ένα νέο σύνθετο τύπο μεταβλητών στην C++,
- να περιγράφουν για ποιον λόγο χρειαζόμαστε τους σύνθετους τύπους μεταβλητών,
- να δηλώνουν μια αναφορά σε μια μεταβλητή στην C++,
- να περιγράφουν για ποιον λόγο χρησιμοποιούμε τις αναφορές,
- να αντιλαμβάνονται τι είναι η διεύθυνση μνήμης μιας μεταβλητής,
- να περιγράφουν με απλά λόγια τι είναι ο δείκτης μνήμης της C++,
- να δημιουργούν έναν δείκτη μνήμης στην γλώσσα προγραμματισμού C++,
- να κάνουν έναν δείκτη μνήμης της C++, να δείχνει σε μια μεταβλητή,
- να περιγράφουν με απλά λόγια τι είναι το πέρασμα ορισμάτων με τιμή,
- να περιγράφουν με απλά λόγια τι είναι το πέρασμα ορισμάτων με αναφορά.

Έννοιες – Λέξεις Κλειδιά

Τεχνικό εγχειρίδιο κατασκευαστή, Στοιχειώδης μεταβλητή, Σύνθετη μεταβλητή, Αναφορά, Σταθερή αναφορά, Δείκτη μνήμης, Διευθυνσιοδότηση, Διεύθυνση μνήμης, Δείκτης μνήμης, Δυναμική διαχείριση μνήμης, Τελεστής έμμεσης διευθυνσιοδότησης, Πέρασμα ορίσματος με τιμή (by value), Πέρασμα ορίσματος με αναφορά (by reference).

Εισαγωγικές Παρατηρήσεις

Κάθε σύστημα σύγχρονης τεχνολογίας, ανεξάρτητα από την υλική ή την άυλη υπόσταση του, (hardware ή software) είναι μια πολύπλοκη σύνθετη κατασκευή η οποία καλύπτει ένα πλήθος λειτουργιών και εμπεριέχει ένα μεγάλο εύρος τεχνικών δυνατοτήτων. Εάν για παράδειγμα μελετούσαμε την κατασκευή μιας σύγχρονης ιπτάμενης μηχανής όπως το αεροσκάφος AIRBUS A320-200, θα αναμέναμε ότι η κατασκευή αυτή θα πρέπει να περιέχει ένα πολύ μεγάλο αριθμό ειδικών θεμάτων που πρέπει να γνωρίζουν οι χρήστες πιλότοι του, με αυτό θα σημαίνει, την ύπαρξη μερικών χιλιάδων σελίδων περιγραφής δυνατοτήτων και τεχνικών θεμάτων.

Για την περίπτωση του πεδίου των γλωσσών προγραμματισμού της επιστήμης της *Πληροφορικής*, μια άυλη κατασκευή όπως, η γλώσσα S.Q.L. (Structured Query Language) που συνοδεύει την βάση δεδομένων Db2, της εταιρίας IBM (της ισχυρότερης σχεσιακής βάσης δεδομένων στον κόσμο) το τεχνικό εγχειρίδιο του κατασκευαστή, που περιγράφει όλες τις δυνατότητες της, εκτείνεται σε ένα βιβλίο 3.271 σελίδων. Αντίστοιχα, το τεχνικό εγχειρίδιο,

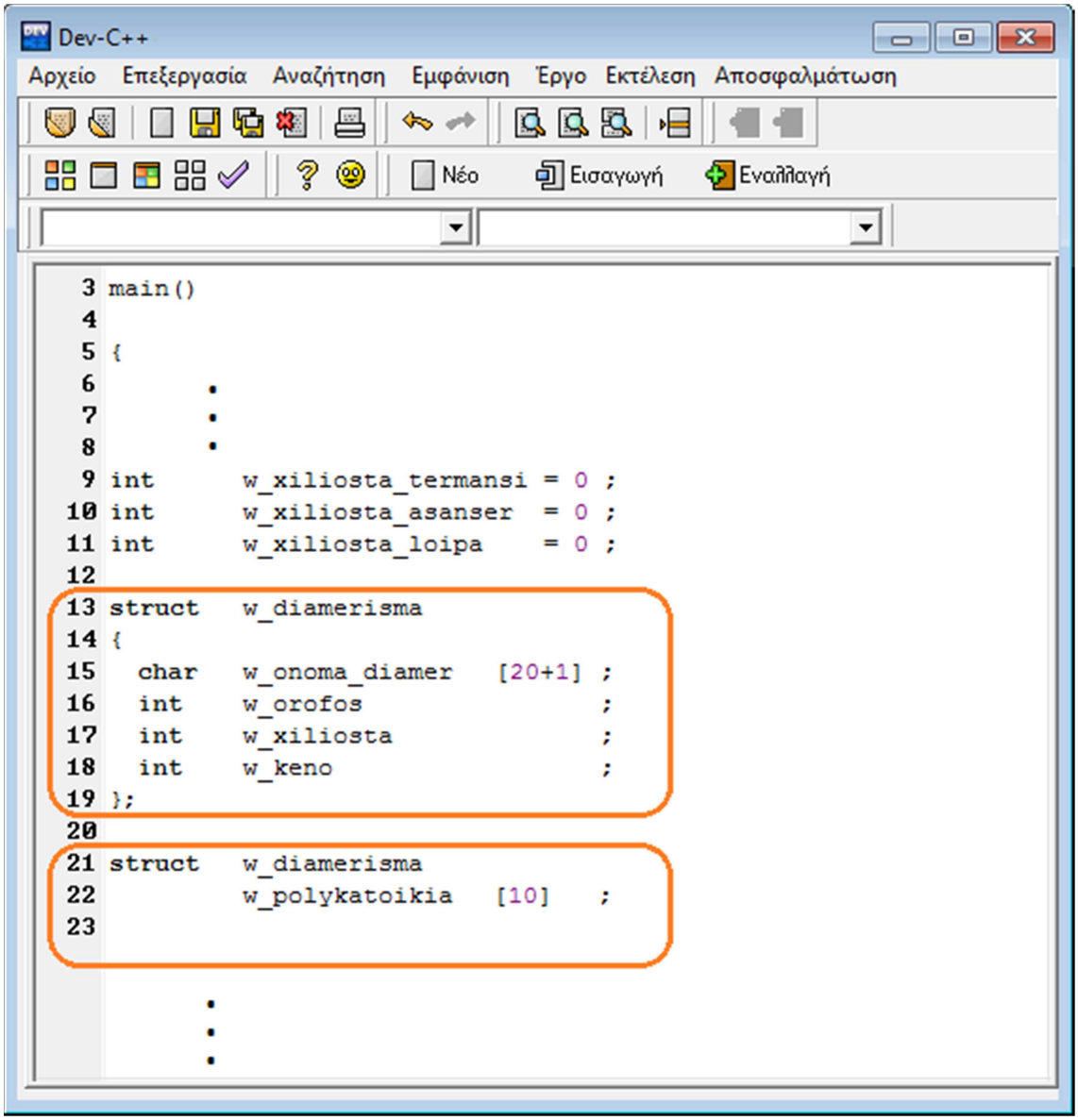
που περιγράφει όλες τις τεχνικές δυνατότητες της γλώσσας C++, κατά το πρότυπο, ISO 2020 που μελετάμε, εκτείνεται σε ένα βιβλίο 1.841 σελίδων.

Από τους αριθμούς των σελίδων και μόνο, αποδεικνύεται ότι μια γλώσσα προγραμματισμού είναι μια εξαιρετικά σύνθετη άυλη κατασκευή που δεν είναι δυνατό να εξαντληθεί η γνώση της, με κάποιον περιορισμένο εκπαιδευτικό χρόνο. Μια σύγχρονη επαγγελματική γλώσσα προγραμματισμού συνεχίζει να μας εκπλήσσει με τα ειδικά της θέματα ακόμα και μετά από πολλά χρόνια από την επαγγελματική της χρήση. Οι νέοι χρήστες «πιλότοι» προγραμματιστές της C++, θα πρέπει να μάθουν να αναζητούν συνεχώς αυτά τα ειδικά της θέματα, να παρακολουθούν συνεχώς τις εξελίξεις της (κάποιο νέο ISO πρότυπο ή κάποια νεότερη έκδοση compiler) και να εμπλουτίζουν την εξειδίκευση τους στην χρήση της γλώσσας. Στο κεφάλαιο αυτό θα παρουσιάσουμε μερικά από αυτά τα ειδικά αλλά απαραίτητα θέματα της γλώσσας προγραμματισμού C++.

4.1. Δημιουργία νέων Τύπων Μεταβλητών

Η γλώσσα C++, όπως και οι περισσότερες γλώσσες προγραμματισμού, παρέχει την δυνατότητα δημιουργίας νέων τύπων μεταβλητών, από συνδυασμό των στοιχειωδών τύπων της. Ο προγραμματιστής μπορεί να δημιουργήσει έναν νέο τύπο μεταβλητής, χρησιμοποιώντας τους στοιχειώδεις τύπους της γλώσσας (**int**, **float**, **char**, κ.λπ.) και να κατασκευάσει νέους σύνθετους τύπους μεταβλητών. Τους νέους αυτούς σύνθετους τύπους, μπορεί στην συνέχεια να τους δηλώσει σαν τύπους μεταβλητών και να τους χρησιμοποιήσει καταλλήλως. Με τον τρόπο αυτό διευκολύνεται η ανάπτυξη ενός προγράμματος. Ένας σύνθετος τύπος μεταβλητών μπορεί επίσης να χρησιμοποιηθεί στην συνέχεια, για την κατασκευή ακόμα ποιο σύνθετων τύπων, που θα αποτελούνται και αυτοί, από συνδυασμούς σύνθετων τύπων. Η δημιουργία ενός νέου σύνθετου τύπου δεδομένων, γίνεται με την εντολή **struct**. Για παράδειγμα, στην επόμενη ανάπτυξη, εκτός από τις μεταβλητές στις γραμμές κώδικα, 9, 10 και 11, που δηλώνονται σαν στοιχειώδεις τύποι **int**, υπάρχει μια σύνθετη δήλωση η οποία αποτελείται από τις στοιχειώδεις δηλώσεις στις γραμμές κώδικα, 15, 16, 17 και 18, οι οποίες θα αποτελούν για την συνέχεια του προγράμματος έναν νέο σύνθετο τύπο δεδομένων (δομή δεδομένων) που ο τύπος της θα ονομάζεται **w_diamerisma** και όταν θα την καλούμε στην πραγματικότητα θα αποτελείται από 4 πεδία. Συνεπώς, εκτός από τους βασικούς «εργοστασιακούς» τύπους

της C++, (**int**, **float**, **double**, **boolean**, **char**, κ.λπ.) θα έχουμε και τον νέο τύπο, **w_diamerisma**. Άρα μπορούμε να τον χρησιμοποιήσουμε στην συνέχεια για να δηλώσουμε νέες μεταβλητές ή να κατασκευάσουμε ακόμα ποιο σύνθετες δομές, όπως γίνεται στις γραμμές κώδικα, 21, 22 και 23. Στην τελευταία αυτή δήλωση, δημιουργείται μια σύνθετη μεταβλητή με το όνομα, **w_polykatoikia**, η οποία είναι ένας πίνακας δέκα θέσεων (λόγω του **[10]**), οι οποίες, κάθε μια από αυτές είναι μεταβλητή τύπου **w_diamerisma**. Συνεπώς, με αυτό τον τρόπο δημιουργήσαμε μια σύνθετη μεταβλητή, την πολυκατοικία, η οποία αποτελείται από 10 διαμερίσματα, που το καθένα από αυτά έχει για στοιχειώδεις μεταβλητές, το όνομα στο διαμέρισμα, τον όροφο, τα χιλιοστά και έναν διακόπτη τον οποίο θα τον χρησιμοποιήσουμε για να δηλώνουμε εάν το διαμέρισμα είναι κλειστό (0=κλειστό, 1=ανοικτό). Από το παράδειγμα αυτό, γίνεται κατανοητό ότι η εντολή **struct**, δημιουργεί απλοποιημένες δομές δεδομένων πριν από τους πινάκων της C++, αλλά και κάθε άλλης δομής δεδομένων.



4.2. Η έννοια της Αναφοράς

Η γλώσσα C++, παρέχει δυνατότητες αναφοράς σε μια μεταβλητή, με το όνομα μιας άλλης μεταβλητής. Η δυνατότητα είναι σαν να αναφερόμαστε στην μεταβλητή με το πρώτο όνομα αλλά χρησιμοποιώντας ένα δεύτερο εναλλακτικό όνομα. Εάν δηλαδή δηλώσουμε την μεταβλητή,

```
int w_pedio;
```

μπορούμε να ορίσουμε και ένα δεύτερο όνομα για την **w_pedio** με την δήλωση,

```
int & w_neo_onoma{w_pedio};
```

το **w_neo_onoma** είναι μια αναφορά και δεν είναι πραγματικά ένα νέο πεδίο στην μνήμη, αλλά «δείχνει» σαν μια αναφορά στο **w_pedio** και άρα θα έχει πρόσβαση στο ίδιο περιεχόμενο του **w_pedio**. Η δήλωση μιας αναφοράς γίνεται με τον ίδιο τύπο ποσότητας που θα πρέπει να ταιριάζει, (π.χ. **int** σε **int**) και στην συνέχεια με το σύμβολο **&**, θα είναι μόνιμη αναφορά μέσα στο πρόγραμμα. Ονομάζεται και σταθερή αναφορά.

Το σύμβολο **&**, όταν εμφανίζεται πριν από μια μεταβλητή, σημαίνει, σε ελεύθερη μετάφραση, κάνε εξαγωγή και χρήση της θέσης μνήμης/διεύθυνσης» της μεταβλητής **w_pedio**, στην μεταβλητή με το νέο όνομα **w_neo_onoma**. Φυσικά πρέπει να έχει προηγηθεί η δήλωση του αυθεντικού πεδίου.

Για να φανεί η σημασία των αναφορών, θα θεωρήσουμε ότι σε ένα software house εργάζονται ταυτόχρονα έλληνες και άγγλοι προγραμματιστές και για να καλυφθούν και οι δυο, οι μεταβλητές στα προγράμματα δηλώνονται με αυτό τον τρόπο,

```
int w_total_amount;
                                        // για τους Άγγλους προγραμματιστές της εταιρίας
int & w_synoliko_poso { w_total_amount };
                    // για τους Έλληνες προγραμματιστές της εταιρίας
```

Ακολουθούν παραδείγματα για τις αναφορές,

```
int pedio;
int & neo_onoma{ pedio };     // σταθερή αναφορά
pedio = 3;                    // εάν δούμε το περιεχόμενο θα είναι, neo_onoma = 3
neo_onoma = 2;                // εάν δούμε το περιεχόμενο θα είναι, pedio = 2
int kati { pedio };           // εάν δούμε το περιεχόμενο θα είναι, kati = 2
int diaf { neo_onoma --};     // εάν δούμε το περιεχόμενο θα είναι, diaf = 2, pedio = 1
int w_a { 8 };                // μη σταθερή αναφορά από, ποσότητα χωρίς όνομα
int w_q { w_a + 12 };         // μη σταθερή αναφορά από το w_a + ποσότητα χωρίς όνομα
```

4.3. Η έννοια του Δείκτη Μνήμης

Κάθε φορά που σε ένα πρόγραμμα C++, δηλώνουμε μια νέα μεταβλητή, δεσμεύουμε και μια αντίστοιχη περιοχή μνήμης στον υπολογιστή. Το μέγεθος αυτή της περιοχής εξαρτάται από τον τύπο της μεταβλητής. Καθώς η μνήμη κάθε υπολογιστή είναι συγκεκριμένη, είναι επίσης οργανωμένη σε τμήματα και άρα είναι προφανές ότι ο υπολογιστής χρησιμοποιεί κάποια μορφής κωδικοποίησης ή μεθοδολογία διευθυνσιοδότησης για να μπορεί να διαχειριστή την μνήμη του. Αυτό σημαίνει ότι η κάθε μεταβλητή που δηλώνεται σε ένα πρόγραμμα συνδέεται προσωρινά και για όσο τρέχει το πρόγραμμα, με έναν συγκεκριμένο αριθμό ο οποίος υποδεικνύει μια θέση μνήμης. Τον κωδικό αυτό αριθμό τον ονομάζουμε *διεύθυνση μνήμης* (memory address). Την διεύθυνσης μνήμης μπορούμε να την χρησιμοποιήσουμε σαν έναν εναλλακτικό τρόπο για να αποκτήσουμε πρόσβαση σε μια μεταβλητή. Αντί δηλαδή να αναφερθούμε με το πραγματικό όνομα στην μεταβλητή, μπορούμε να το κάνουμε έμμεσα μέσω της διεύθυνσης μνήμη της. Οι γλώσσες προγραμματισμού μας επιτρέπουν να βρούμε την διεύθυνση μνήμης που «πατάει» μια μεταβλητή, και αφού την βρούμε, να τοποθετήσουμε αυτόν τον κωδικό διεύθυνσης, μέσα σε μια άλλη μεταβλητή που ονομάζεται, *δείκτης μνήμης* (nodes) και να αναφερόμαστε στην μεταβλητή μέσω του δείκτη μνήμης της. Η διαδικασία

αυτή ονομάζεται, *δυναμική διαχείριση μνήμης* με δείκτες μνήμης. Και στην «αργκό» της γλώσσας C++, λέμε ότι, «ο δείκτης μνήμης δείχνει στην μεταβλητή».

Η δήλωση ενός δείκτη μνήμης (pointer) στην C++, γίνεται με την εντολή,

```
int * w_deiktis ;
```

Οι δείκτες μνήμης είναι τύποι δεδομένων που απλά περιέχουν διευθύνσεις. Από την στιγμή που δηλώσουμε έναν δείκτη μνήμης για να έχει νόημα η υπόσταση του, θα πρέπει να τον συνδέσουμε με μια κανονική μεταβλητή ή όπως ήδη αναφέρθηκε, «να βάλουμε τον δείκτη μνήμης να δείχνει σε μια μεταβλητή». Αυτό γίνεται με τις αρχικές δηλώσεις,

```
int w_metavliti = 234 ;
int * w_deiktis;
```

και την εντολή ανάθεσης,

```
w_deiktis = & w_metavliti ;
```

η οποία βάζει τον δείκτη **w_deiktis**, να δείχνει στην κανονική μεταβλητή **w_metavliti**. Το σύμβολο **&**, που εμφανίζεται πριν από την **w_metavliti**, σημαίνει σε ελεύθερη μετάφραση, «κάνε εξαγωγή και χρήση της θέσης μνήμης/διεύθυνσης» της μεταβλητής **w_metavliti**, επάνω στον δείκτη με το όνομα **w_deiktis**. Συνεπώς, εάν η **w_metavliti**, έχει για περιεχόμενο το **234** και υποθέσουμε ότι τοποθετήθηκε στην θέση μνήμης 1087, τότε εάν τυπώσουμε στην οθόνη μας, το περιεχόμενο της **w_deiktis**, δεν θα δούμε το **234**, αλλά το 1087. Το σύμβολο *****, που εμφανίζεται πριν από την **w_deiktis**, λειτουργεί σαν ένας ειδικός τελεστής στην γλώσσα C++, και ονομάζεται, *τελεστής έμμεσης διευθυνσιοδότησης*. Το σύμβολο *****, μπορεί να το δούμε να χρησιμοποιείται με δυο τρόπους, α) κατά την δήλωση μια μεταβλητής δείκτη και β) να το έχουμε μπροστά από ένα δείκτη και άρα να αναφερόμαστε στην μεταβλητή που δείχνει αυτός ο δείκτης. Οι δείκτες μνήμης στην C++, χρησιμοποιούνται για την διαπέραση των πινάκων αλλά και για πέρασμα ορισμάτων σε μια συνάρτηση. Με την χρήση τους μπορούμε να

επηρεάσουμε την συμπεριφορά της εμβέλειας των μεταβλητών. Για παράδειγμα, όταν σε ένα πρόγραμμα C++, περνάμε σε μια εσωτερική συνάρτηση, ορίσματα μέσω μεταβλητών, οι αλλαγές που πιθανά γίνονται επάνω σε αυτές τις μεταβλητές-ορίσματα, από τον κώδικα που υπάρχει μέσα στην εσωτερική συνάρτηση, δεν θα εξακολουθήσουν να υφίστανται όταν ο έλεγχος του προγράμματος (η «μπίλια») επιστρέψει πίσω στο σημείο που καλέσαμε την συνάρτηση/ρουτίνα. Οι ενδεχόμενες αλλαγές δηλαδή, δεν θα γυρίσουν πίσω, προς τις μεταβλητές που βρίσκονταν στην είσοδο της συνάρτησης (τα ορίσματα εισόδου).
Το πέρασμα που κάνουμε με την εντολή,

```
void my_routina ( int w_metavliti_a, int w_metavliti_b )
        {
          … σώμα εντολών που θα εκτελεστούν όταν κληθεί η συνάρτηση
        } ;
```

στην πραγματικότητα, περνά μόνο το περιεχόμενο των μεταβλητών και όχι ολόκληρες τις μεταβλητές και το κάνει για τοπική χρήση του περιεχομένου μέσα στην συνάρτηση. Αυτό στην «αργκό» της C++, ονομάζεται, «πέρασμα ορίσματος σε συνάρτηση με τιμή - by value». Συνεπώς, τίθεται το ερώτημα, μπορούμε να περάσουμε προς μια συνάρτηση, όχι μόνο το περιεχόμενο μιας μεταβλητής (by value) αλλά ολόκληρη την μεταβλητή. Η γλώσσα C++, μπορεί να το κάνει αυτό με κατάλληλη τεχνική που στην «αργκό» της ονομάζεται, «πέρασμα ορίσματος σε συνάρτηση με αναφορά - by reference».

Στην περίπτωση αυτή περνάμε τις μεταβλητές, με τους δείκτες μνήμης τους,

```
void my_routina ( int * w_deiktis_metavliti_a, int * w_deiktis_metavliti_b )
        {
          … σώμα εντολών που θα εκτελεστούν όταν κληθεί η συνάρτηση
        } ;
```

Στο πέρασμα ορίσματος σε συνάρτηση με τιμή - by value, η κλήση της συνάρτησης my_routina, κάπου μέσα στο πρόγραμμα θα γίνει με την εντολή,

```
my_routina ( w_metavliti_a, w_metavliti_b ) ;
```

ενώ στο πέρασμα ορίσματος σε συνάρτηση με αναφορά - by reference, η κλήση της συνάρτησης **my_routina**, κάπου μέσα στο πρόγραμμα θα πρέπει να γίνει με την εντολή,

```
my_routina ( & w_metavliti_a, & w_metavliti_b ) ;
```

Αυτό, στην C++, είναι μια πρακτική ταυτόχρονης χρήσης των αναφορών μαζί με τους δείκτες μνήμης.

Σύνοψη κεφαλαίου

Στο τέταρτο κεφάλαιο περιγράφηκαν κυρίως τα ειδικά θέματα των αναφορών και των δεικτών μνήμης της γλώσσας C++. Επιπλέον, παρουσιάστηκε και η δυνατότητα δημιουργίας νέων τύπων δεδομένων τα οποία μπορούν να χρησιμοποιηθούν για την κατασκευή σύνθετων τύπων μεταβλητών.

Ερωτήσεις αξιολόγησης

Ερώτηση-1: Περιγράψτε τι είναι ένας σύνθετος τύπος μεταβλητών στην C++;

Ερώτηση-2: Περιγράψτε ένα παράδειγμα σύνθετου τύπου μεταβλητών;

Ερώτηση-3: Περιγράψτε τι είναι η αναφορά στην C++ και δώστε ένα παράδειγμα;

Ερώτηση-4: Περιγράψτε με απλά λόγια, τι είναι ο δείκτης μνήμης στην C++;

Ερώτηση-5: Περιγράψτε, με ποιο τρόπο σχετίζεται μια μεταβλητή ενός προγράμματος C++, με μια διεύθυνση μνήμης στον υπολογιστή;

Ερώτηση-6: Είναι σωστή, η δήλωση **int * w_my_pedio** ;

Ερώτηση-7: Είναι σωστή, η δήλωση **int & w_my_pedio** ;

Ερώτηση-8: Είναι σωστή, η δήλωση **int * w_xxx_pedio { & w_yyy-pedio }** ;

Ερώτηση-9: Είναι σωστή, η δήλωση **int & w_xxx_pedio { * w_yyy-pedio }** ;

Ερώτηση-10: Περιγράψτε, ποια είναι η διαφορά, περάσματος ορίσματος σε συνάρτηση με τιμή, σε σχέση με το πέρασμα ορίσματος σε συνάρτηση με αναφορά;

Απαντήσεις ερωτήσεων αξιολόγησης

Απάντηση-1: Συμβουλευτείτε την ενότητα 4.1. του κεφαλαίου.

Απάντηση-2: Συμβουλευτείτε την ενότητα 4.1. του κεφαλαίου.

Απάντηση-3: Συμβουλευτείτε την ενότητα 4.2. του κεφαλαίου.

Απάντηση-4: Συμβουλευτείτε την ενότητα 4.3. του κεφαλαίου.

Απάντηση-5: Συμβουλευτείτε την ενότητα 4.3. του κεφαλαίου.

Απάντηση-6: Ναι, είναι σωστό.

Απάντηση-7: Ναι, είναι σωστό.

Απάντηση-8: Ναι, είναι σωστό.

Απάντηση-9: Όχι, είναι λάθος.

Απάντηση-10: Συμβουλευτείτε την ενότητα 4.3. του κεφαλαίου.

5. ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΕΙΑ – ΤΕΧΝΟΤΡΟΠΙΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Σκοπός και επιμέρους στόχοι

Η σχεδίαση ενός λογισμικού συστήματος μπορεί να υλοποιηθεί με διάφορες μεθοδολογίες ανάπτυξης. Οι μεθοδολογίες αυτές κατηγοριοποιούνται συνήθως, σε διαδικασιακές και σε διαδικασίες που είναι προσανατολισμένες προς την έννοια των *αντικειμένων*. Αυτή η δεύτερη γενική κατηγοριοποίηση που ονομάζεται *αντικειμενοστρέφεια* (object-oriented) αφορά και τις γλώσσες προγραμματισμού και την C++. Στο κεφάλαιο αυτό γίνεται μια περιεκτική αναφορά στα τεχνικά χαρακτηριστικά της αντικειμενοστρέφειας και των ιδιόμορφων εννοιών που περιέχει.

Προσδοκώμενα αποτελέσματα

Με την μελέτη του κεφαλαίου οι εκπαιδευόμενοι θα μπορούν,

- να περιγράφουν βασικές έννοιες της αντικειμενοστρέφειας,
- να ορίσουν την έννοια της κλάσης και να περιγράψουν τι σημαίνει αντικείμενο μιας κλάσης,
- να ορίσουν με απλά λόγια μια κλάση και μια υποκλάση,
- να δώσουν παραδείγματα κλάσεων και να τους δηλώσουν αντικείμενα,
- να εξηγήσουν τι σημαίνει μέθοδος,
- να περιγράψουν με απλά λόγια τι είναι ένα στιγμιότυπο,
- να περιγράψουν με απλά λόγια την έννοια της κληρονομικότητας στην αντικειμενοστρέφεια,
- να αναγνωρίσουν πλεονεκτήματα στην αντικειμενοστρέφεια,
- να περιγράψουν μειονεκτήματα και πλεονεκτήματα ανάμεσα στις βασικές διαδικασιακές και τις αντικειμενοστραφείς γλώσσες προγραμματισμού,
- να αναφέρουν με βασικά απλά λόγια την σημασία των βάσεων δεδομένων (Data Bases) και της ενσωματωμένης γλώσσας S.Q.L.

Έννοιες – Λέξεις Κλειδιά

Αντικείμενο, Αντικειμενοστραφής σχεδίαση, Αντικειμενοστραφής τεχνολογία προγραμματισμού, Unified Modeling Language, Κλάση, Υπέρ-κλάση, Υπό-κλάση, Τακτοποίηση αντικειμένων, Μέθοδος, Στιγμιότυπο, Κληρονομικότητα, Ενθυλάκωση, Αφαιρετικότητα, Υπερφόρτωση, Πολυμορφισμός, Structured Query Language.

Εισαγωγικές Παρατηρήσεις

Η ακολουθιακή ή διαδικασιακή προσέγγιση προγραμματισμού και η δομημένη ανάλυση, παρουσιάστηκε αναλυτικά στα δυο πρώτα συγγράμματα της σειράς, *Προγραμματισμός Ηλεκτρονικών Υπολογιστών & Μηχανών*, της ΑCTA αλλά και σε όλα τα υπόλοιπα κεφάλαια του 3ου συγγράμματος. Αυτό το παρεχόμενο εκπαιδευτικό υλικό περιελάμβανε όλα τα απαραίτητα βασικά γνωσιακά στοιχεία για να δημιουργήσει στον αναγνώστη εκπαιδευόμενο, το κατάλληλο, θεωρητικό και πρακτικό υπόβαθρο γνώσης για την γλώσσα προγραμματισμού C++, την ακολουθιακή προσέγγιση και την δομημένη ανάλυση και σχεδίαση. Στο συγκεκριμένο κεφάλαιο του 3ου συγγράμματος οι εκπαιδευόμενοι εισάγονται σε μια διαφορετική τεχνοτροπία ανάπτυξης και προγραμματισμού που ονομάζεται, *αντικειμενοστραφής σχεδίαση* και *αντικειμενοστραφής τεχνολογία προγραμματισμού*. Η αντικειμενοστραφής προσέγγιση προγραμματισμού περιλαμβάνει ένα ιδιαίτερα εκτεταμένο πεδίο γνώσης το οποίο δεν είναι δυνατό να το προσεγγίσουμε μέσα από τον περιορισμένο χώρο ενός κεφαλαίου ενός συγγράμματος. Παρόλα αυτά, η εισαγωγική παράθεση των βασικών του εννοιών και η επεξήγηση τους, θα δώσει στους αναγνώστες όλα τα κατάλληλα εφόδια για την συνέχιση της μελέτης τους, στις σχετικές βιβλιογραφικές παραπομπές.

Ο όρος, *αντικειμενοστραφής προγραμματισμός,* προέρχεται από τον αγγλικό όρο, Object Oriented Programming (O.O.P.), που στα ελληνικά θα μπορούσε να μεταφραστεί απλοποιημένα, από έναν Πληροφορικό, σαν «προγραμματισμός, προσανατολισμένος γύρω από τα αντικείμενα». Και βέβαια η παράθεση της συγκεκριμένης έκφρασης, από μόνη της δεν επαρκεί, καθώς δεν γίνεται κατανοητή ειδικά από τους νέους προγραμματιστές και θα πρέπει να επεξηγηθεί, μέσω κάποιας εξοικείωσης από προϋπάρχουσα γνώση και εμπειρία από τον κλασικό διαδικασιακό προγραμματισμό. Ο όρος, *αντικειμενοστραφής,* εμφανίζεται στην ελληνική βιβλιογραφία και σαν *αντικειμενοστρεφής*, με την ίδια ακριβώς σημασία.

Η έκφραση, Object Oriented Programming, στην πραγματικότητα σημαίνει, μια φιλοσοφία αντίληψης του προγραμματισμού, με ένα διαφορετικό πνεύμα, ή παρατήρησης μέσα από ένα διαφορετικό πρίσμα. Για παράδειγμα, αυτό που σε μια άλλη τεχνοτροπία θα ονομαζόταν ρουτίνα, στην O.O.P. φιλοσοφία, θα ονομάζεται, *μέθοδος* (method). Ο αντικειμενοστραφής προγραμματισμός, δεν είναι μια νέα τεχνοτροπία, όπως λανθασμένα προβάλλεται, αλλά

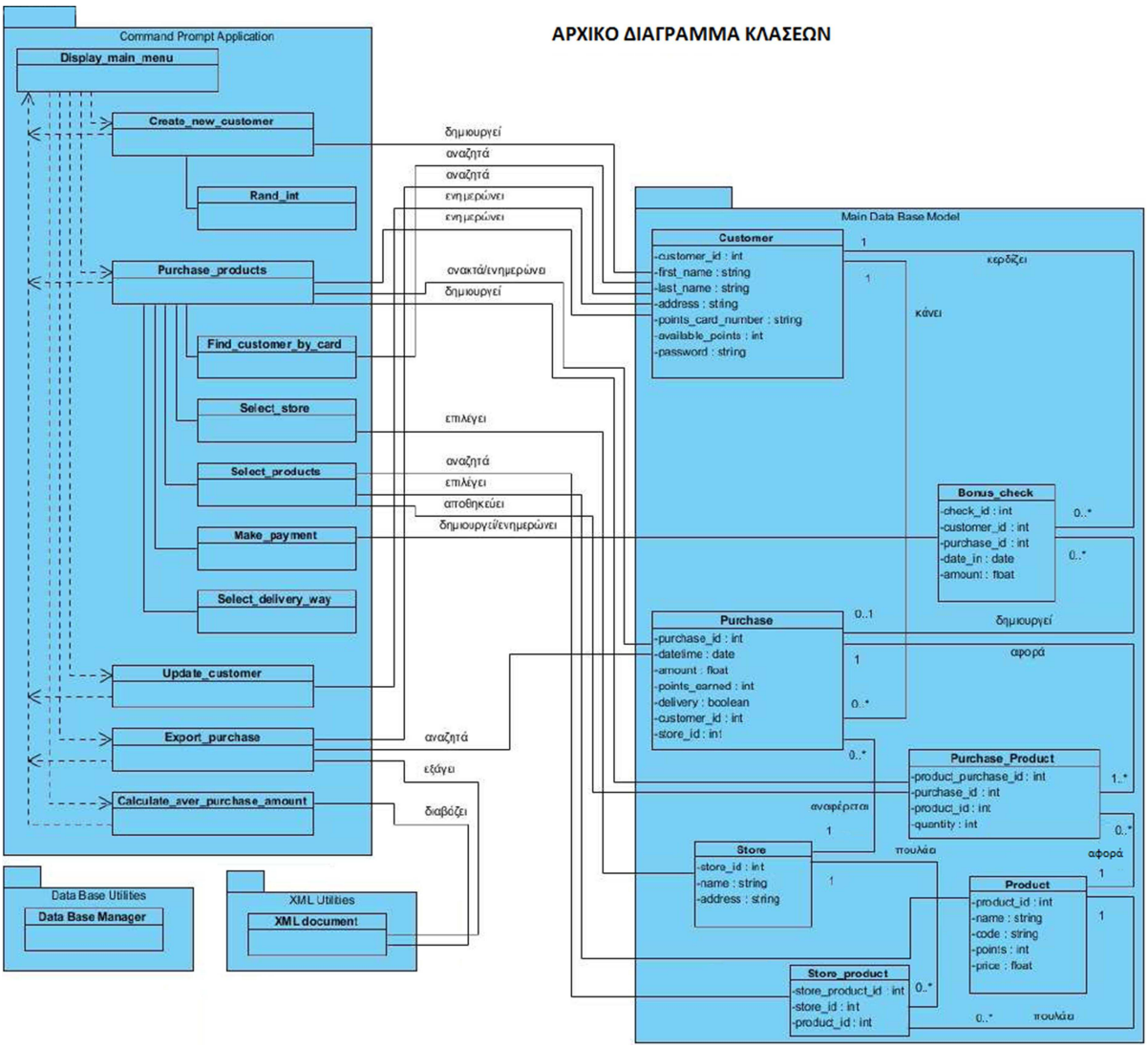
υπάρχει και εξελίσσεται σαν παράλληλη τεχνοτροπία προγραμματισμού ήδη από την δεκαετία του 1980, με ρίζες από την δεκαετία του 1960. Συνεπώς ο αντικειμενοστραφής προγραμματισμός δεν είναι σε καμία περίπτωση το εκπληκτικό νέο περιβάλλον ανάπτυξης, αλλά ένα διαθέσιμο επιπλέον εργαλείο στον τομέα της *Τεχνολογίας Λογισμικού*, της επιστήμης της *Πληροφορικής*.

Δανειζόμενοι την επεξήγηση του όρου, από ένα από τα σημαντικότερα πρόσωπα του χώρου, τον Grady Booch, το Object Oriented Programming είναι, η μεθοδολογία να συγγράφουμε κώδικα (συναρτήσεις, ρουτίνες, ή μεθόδους), με τέτοιο τρόπο ώστε, να συνδυάζουμε λειτουργικές εντολές προγραμματισμού (ή διαδικασίες) αλλά και δεδομένα (ή πληροφορίες) και όταν το πετυχαίνουμε αυτό, να αποθηκεύουμε αυτή την κατασκευή, σε μια δομική μορφή, που θα την ονομάζουμε object, (δηλαδή, αντικείμενο). Τα αντικείμενα αυτά μπορούμε επιπλέον να τα τακτοποιούμε σε συλλογές αντικειμένων, εφαρμόζοντας μια ιεραρχική δομή σχέσεων όπου η κάθε συλλογή θα μπορεί να κληρονομεί (ή να κληρονομείται) λειτουργικότητα, προς (ή από) μια άλλη συλλογή αντικειμένων. Και από αυτόν τον ορισμό προήλθε η έκφραση *αντικειμενοστραφής προγραμματισμός,* που σε απλά ελληνικά θα μπορούσε να σημαίνει*, φτιάξε κώδικα που χρησιμοποιεί κατάλληλες διεπαφές με άλλους κώδικες, επιτελεί μια συγκεκριμένη λειτουργία, μετασχηματίζει δεδομένα και φύλαξε τον να υπάρχει και όταν τον χρειαστείς κάλεσε τον.*

Η *αντικειμενοστρέφεια,* όπως απλοποιημένα αποκαλείται, από τους προγραμματιστές θεωρείται, για ένα λογισμικό, μια συλλογή από αντικείμενα (object), το καθένα από τα οποία περιέχει διαδικασίες και δεδομένα. Η κλήση ενός αντικειμένου, από ένα άλλο αντικείμενο επιστρέφει στο αντικείμενο που έκανε την κλήση, ένα είδος «υπηρεσιών» με την μορφή δεδομένων αλλά και κώδικα. Αυτή η διαδοχική παροχή κατώτερων «υπηρεσιών» προς άλλες «υπηρεσίες» ανώτερου επιπέδου, δημιουργεί τελικά, την αποτελεσματικότητα μιας εφαρμογής προς ένα εξωτερικό περιβάλλον.

Η αντικειμενοστρέφεια, εισάγει πολλές έννοιες ιδιαίτερης σημασιολογίας και ειδικές προγραμματιστικές τεχνικές που δημιουργούν σύγχυση και αποπροσανατολίζουν. Αυτή η κατάσταση σύγχυσης παρατηρείται και στο επίπεδο της εκπαίδευσης όπου παραποιείται

ακόμα και ο ορισμός της αντικειμενοστρέφειας, ενώ είναι συνηθισμένο ατυχές φαινόμενο να παρατίθενται άστοχα παραδείγματα, για αντικείμενα «σκύλους», μεθόδους που «γαβγίζουν» και τμήματα σπουδαστών που το σύνολο του πληθυσμού τους αποτυγχάνει στις εξετάσεις. Αυτά τα προβλήματα της αντικειμενοστρέφειας διαφαίνεται να επιλύονται διαμέσου του τομέα της τεχνολογίας λογισμικού και των συστημάτων σχεδιασμού U.M.L. (Unified Modeling Language), της γλώσσας που επίσης δημιουργήθηκε από τον Grady Booch, του προσώπου από το οποίο δανειστήκαμε την επεξήγηση για τον όρο, Ο.Ο.Ρ.



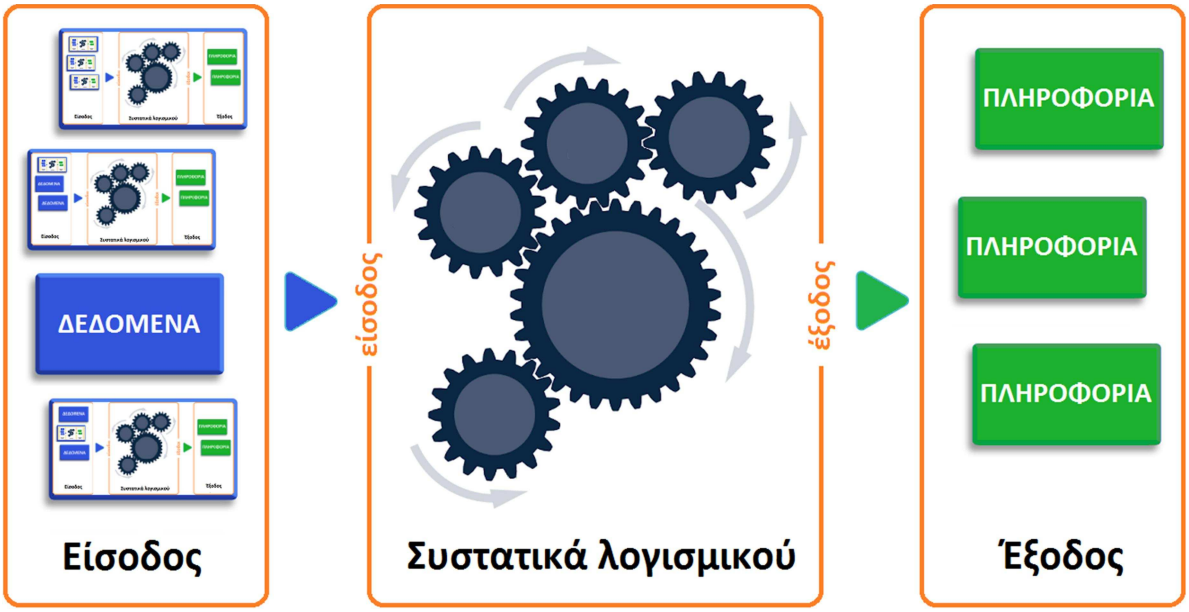
Διάγραμμα κλάσεων παραγόμενο, από το U.M.L. περιβάλλον, Visual Paradigm.

Η U.M.L. είναι μια τυποποιημένη «γλώσσα» που περιγράφει τα συστήματα λογισμικού με ειδικά σχεδιαγράμματα και βασίζεται στις αρχές και τα πρότυπα της αντικειμενοστρέφειας. Η

U.M.L. ως παγκόσμιο πρότυπο παράστασης λογισμικού, επιτρέπει την οπτική αναπαράσταση, της ανάλυσης, της καταγραφής των απαιτήσεων, της σχεδίασης, της υλοποίησης και της τεκμηρίωσης ενός πληροφοριακού συστήματος, με τρόπο κατανοητό στον καθένα που εμπλέκεται στην ανάπτυξη, χρήση και υποστήριξη ενός λογισμικού (πελάτη, αναλυτή, μηχανικό λογισμικού, προγραμματιστή, χειριστή, κ.λπ.).

5.1. Αντικείμενα – object

Ο όρος *αντικείμενο* (object), όπως εισαγωγικά επεξηγήθηκε στην αρχή του κεφαλαίου αφορά την περιγραφή μιας δομικής μορφής κώδικα, η οποία βασίζεται σε δεδομένα (ή «στοχεύει» τα δεδομένα), τα οποία μέσω κάποιων λειτουργικών εντολών προγραμματισμού (διαδικασίες ή υπολογισμούς) τα μετασχηματίζει σε μετα-δεδομένα, την χρονική στιγμή που θα κληθεί. Στην αντικειμενοστρέφεια, ένα object, θα ανταποκριθεί μια συγκεκριμένη χρονική στιγμή σε μια κατάλληλη κλήση, από ένα άλλο κατάλληλο αντικείμενο (object), το οποίο και αυτό μπορεί να έχει κληθεί με την σειρά του από ένα άλλο ανώτερο object. Αντικείμενο (object), μπορεί να χαρακτηριστεί και μια συλλογή δεδομένων, ακόμα και πρωτογενούς τύπου, που μετασχηματίζεται ή δεν μετασχηματίζεται και απλά χρησιμοποιείται. Για την εκπαίδευση της πληροφορικής, η δυσκολία κατανόησης του όρου object, θεωρείται δεδομένη και απαιτεί χρόνο και εμπειρίες. Μια κατάλληλη όμως σχηματική απεικόνιση μπορεί να λειτουργήσει υποβοηθητικά. Η επόμενη εικόνα αποτυπώνει -αφαιρετικά- για την αντικειμενοστρέφεια, τις έννοιες των objects και της *ενθυλάκωσης* (ecapsulation) και διαφοροποιείται εννοιολογικά με την αντίστοιχη απεικόνιση στον ακολουθιακό προγραμματισμό.



5.2. Κλάσεις – class

Ο όρος *κλάση* (class), προέρχεται από την αγγλική συντομογραφία class, του όρου classification, το οποίο μπορεί να αποδοθεί καλύτερα, όχι στην ελληνική λέξη «κλάση», ούτε στην ελληνική λέξη «ταξινόμηση» αλλά στις ελληνικές λέξεις, «κατάταξη», «ένταξη» ή «τακτοποίηση». Ο όρος κλάση, δημιουργεί μια από τις μεγαλύτερες συγχύσεις στην αντικειμενοστρέφεια καθώς δεν περιγράφει την «ταξινόμηση των πραγμάτων» αλλά την «τακτοποίηση των πραγμάτων» σε τάξεις (σε κατηγορίες). Συγκεκριμένα, με την είσοδο μας σε μια αίθουσα διδασκαλίας, μπορούμε να παρατηρήσουμε, «τάξεις πραγμάτων», όπως, οι σπουδαστές, τα έπιπλα, τα βιβλία, τα μαθήματα, τα εποπτικά μέσα αλλά και κάθε άλλο αντικείμενο (υλικό ή άυλο) για το οποίο θα του άξιζε η προσοχή μας. Συνεπώς, εάν μας ζητείτο να κατασκευάσουμε ένα λογισμικό σε μια αντικειμενοστραφή γλώσσα, που να υποστηρίζει την λειτουργία των εκπαιδευτικών δομών της ΑΣΤΑ, θα χρειαζόταν οπωσδήποτε να δημιουργήσουμε τις κλάσεις, ΣΠΟΥΔΑΣΤΕΣ, ΕΠΙΠΛΑ, ΒΙΒΛΙΑ, ΜΑΘΗΜΑΤΑ και ΕΠΟΠΤΙΚΑ ΜΕΣΑ. Ακολούθως, θα μπορούσαμε να κατατάξουμε (και όχι να ταξινομήσουμε) τα object, ΤΗΛΕΟΡΑΣΗ, ΠΙΝΑΚΑΣ και ΠΟΛΥΜΕΤΡΟ στην κλάση των ΕΠΟΠΤΙΚΩΝ ΜΕΣΩΝ. Τα 3 συγγράμματα της σειράς, *Προγραμματισμός Ηλεκτρονικών Υπολογιστών & Μηχανών*, της ΑΣΤΑ, στην κλάση των ΒΙΒΛΙΩΝ. Τον σπουδαστή ΕΛΕΥΘΕΡΙΟΥ και τις σπουδάστριες ΝΙΚΟΥ και ΔΗΜΟΥ στην κλάση ΣΠΟΥΔΑΣΤΕΣ, αλλά να δημιουργήσουμε και μια άλλη class για τα άυλα object όπως η ΠΛΗΡΟΦΟΡΙΚΗ, η ΦΥΣΙΚΗ και η ΗΛΕΚΤΡΟΛΟΓΙΑ που ανήκουν φυσικά στην

κλάση ΜΑΘΗΜΑΤΑ. Όμως η αντικειμενοστρέφεια θα μας επέτρεπε να δημιουργήσουμε και μια άλλη κλάση, την οποία να μπορούσαμε να ονομάσουμε ΣΥΝΑΡΤΗΣΕΙΣ-ΕΛΕΓΧΟΥ και μέσα της να τοποθετήσουμε σαν αντικείμενα, σχετικές μεθόδους (και όχι αντικείμενα του φυσικού κόσμου) που διαχειρίζονται γενικότερα στο λογισμικό μας, ημερομηνίες και ώρες μαθημάτων. Οι κλάσεις συνεπώς περιέχουν και όλες τις απαραίτητες μεθόδους που έχουν σχέση με την κλάση και τα στοιχεία της.

5.3. Στιγμιότυπα – instances

Έχοντας περιγράψει από την προηγούμενη ενότητα τον βασικό όρο της κλάσης και χρησιμοποιώντας για παράδειγμα την κατασκευή ενός λογισμικού με τις κλάσεις ΣΠΟΥΔΑΣΤΕΣ, ΕΠΙΠΛΑ, ΒΙΒΛΙΑ, ΜΑΘΗΜΑΤΑ, ΕΠΟΠΤΙΚΑ ΜΕΣΑ και ΣΥΝΑΡΤΗΣΕΙΣ-ΕΛΕΓΧΟΥ, θα συναντήσουμε στην αντικειμενοστρέφεια και τον βασικό όρο *στιγμιότυπο* (instance).

Ο όρος αφορά τα αντικείμενα, ΤΗΛΕΟΡΑΣΗ, ΕΛΕΥΘΕΡΙΟΥ, ΠΛΗΡΟΦΟΡΙΚΗ αλλά και όλα τα υπόλοιπα όμοια αντικείμενα, τα οποία, το καθένα είναι ένα στιγμιότυπο της κλάσης στην οποία ανήκει. Επιπλέον, ένα στιγμιότυπο, μπορεί να διαφοροποιείται ανάλογα με το περιβάλλον που μπορεί να το επηρεάζει. Για παράδειγμα, το μάθημα ΠΛΗΡΟΦΟΡΙΚΗ, το οποίο διδάσκεται σήμερα στις 14.00, εάν το παρατηρήσουμε στις 11.00 το πρωί είναι ένα στιγμιότυπο Α, ενός μαθήματος που δεν έχει πραγματοποιηθεί, αλλά στις 18.00 το απόγευμα, είναι πάλι το ίδιο μάθημα (το ίδιο αντικείμενο) που έχει όμως πραγματοποιηθεί και άρα αποτυπώνεται, αφού επηρεάζεται διαφορετικά από το περιβάλλον του (τον χρόνο) σαν ένα άλλο στιγμιότυπο Β, του ίδιου αντικειμένου (του μαθήματος ΠΛΗΡΟΦΟΡΙΚΗΣ) της κλάσης ΜΑΘΗΜΑΤΑ.

5.4. Μέθοδοι – methods

Με τον όρο *μέθοδος* (method), περιγράφουμε δομικά στοιχεία κώδικα ή ποιο απλά, σειρά από εντολές προγραμματισμού. Την ίδια ακριβώς σημασία αποτυπώνουν οι γνωστοί όροι, ρουτίνα, υπορουτίνα και συνάρτηση. Μέσα σε μια method, οι εντολές προγραμματισμού εκτελούνται ακολουθιακά, με την γνωστή λογική της «μπίλιας» και των πιθανών εντολών ανακατεύθυνσης. Μια μέθοδος μπορεί να κληθεί από μια άλλη μέθοδο ή να καλεί με την σειρά της άλλες μεθόδους. Και στην περίπτωση της αντικειμενοστρέφειας έχουμε την κυρίαρχη, αρχική **main**

μέθοδο. Οι μέθοδοι συνοδεύονται με τις κλασικές έννοιες, των ορισμάτων εισόδου, των ορισμάτων εξόδου, των επιστροφών, αλλά επίσης και με νέες έννοιες, όπως, κατασκευάστρια μέθοδος, μέθοδος getter, μέθοδος setter, κ.λπ.

5.5. Κληρονομικότητα – inheritance

Ο όρος *κληρονομικότητα* (inheritance), είναι ίσως η σπουδαιότερη τεχνική ευκολία της τεχνοτροπίας Ο.Ο.Ρ. Η κληρονομικότητα βασίζεται στην σχέση γενίκευσης – εξειδίκευσης μεταξύ κλάσεων και συνδέεται άμεσα με τον όρο *υπό-κλάσεις* (subclass). Αυτό που γίνεται με την κληρονομικότητα στην πραγματικότητα είναι ότι, οι κλάσεις διασυνδέονται μεταξύ τους, με έναν κατάλληλο τρόπο, ώστε δομικά στοιχεία τα οποία μπορεί να είναι κοινά, να μετακινούνται σε κλάσεις «απογόνους» και να αποφεύγεται η επανάληψη όμοιου κώδικα, όμοιων μεταβλητών και όμοιων συστατικών λογισμικών (συναρτήσεων ή μεθόδων για την Ο.Ο.Ρ.). Αυτό μπορεί να είναι ιδιαίτερα ωφέλιμο καθώς απλοποιεί την φάση της ανάπτυξης ενός αξιόλογου λογισμικού αλλά μπορεί να είναι πολύ περισσότερο ωφέλιμο κατά την συνεχή φάση της συντήρηση του. Είναι ευκολότερο να συντηρήσεις μια εφαρμογή με 50 κλάσεις, από μια εφαρμογή με 80 κλάσεις όπου πολλά πράγματα επαναλαμβάνονται.

Για να γίνει αυτό κατανοητό και στο επίπεδο της δεδομένης απειρίας των εκπαιδευομένων στην Ο.Ο.Ρ. αλλά και της ανάγκης να λάβουν ένα ικανό γνωσιακό υπόβαθρο για την περαιτέρω μελέτη του θέματος της κληρονομικότητας θα χρησιμοποιήσουμε για παράδειγμα μια εφαρμογή διαχείρισης των ΕΛ.ΤΑ., τα οποία μας εξυπηρετούν με τις ταχυδρομικές τους υπηρεσίες. Σε αυτό το σύστημα θα μπορούσαν να υπάρχουν πολλές δεκάδες κλάσεις αλλά εμείς θα επικεντρωθούμε σε κάποιες κλάσεις οι οποίες είναι αναγκαίες για την διαχείριση των χάρτινων κιβωτίων (πακέτων), σε ότι αφορά την καταμέτρηση του όγκου τους, έτσι ώστε να μπορούν τα ΕΛ.ΤΑ. να υπολογίζουν τις αναγκαίες και διαθέσιμες παλέτες φόρτωσης. Ας υποθέσουμε τώρα ότι τα χάρτινα κιβώτια τα οποία αναλαμβάνουν τα ΕΛ.ΤΑ. να διαχειριστούν, διαχωρίζονται σε δυο βασικές μορφές, α) τα ορθογώνια κουτιά και β) τα κυλινδρικά κουτιά ή κυλίνδρους. Στην περίπτωση αυτή μπορούμε να υποθέσουμε ότι θα χρειαστούμε δυο διαφορετικές κλάσεις, μια για τα κιβώτια και μια για τους κυλίνδρους. Εφόσον προσπαθούμε να αναπτύξουμε μια εφαρμογή για την διαχείριση αυτών των αντικειμένων και για να καταφέρουμε να υπολογίζουμε τους όγκους τους, θα χρειαζόμασταν κατά περίπτωση,

α) για τα κιβώτια,

α.1) το πλάτος,

α.2) το μήκος,

α.3) το ύψος,

β) για τους κυλίνδρους,

β.1) το ύψος,

β.2) την ακτίνα,

και για την ολοκλήρωση των δυο κλάσεων, επίσης,

γ.1) την περιγραφή του κουτιού,

γ.2) το χρώμα του κουτιού,

γ.3) το υλικό του κουτιού.

Στην περίπτωση του υπολογισμού του όγκου των κιβωτίων ο αλγόριθμος αποτυπώνεται με την πράξη, *πλάτος x μήκος x ύψος*, ενώ για τους κυλίνδρους, με την πράξη, *ακτίνα*^{2} *x π x ύψος*. Συνεπώς, οι δύο κλάσεις, για την διαχείριση των αντικειμένων αυτών, μαζί με τα επιπλέον περιγραφικά τους στοιχεία (περιγραφή, χρώμα, κ.λπ.) θα αποτυπώνονταν στην σχεδίαση μας ως εξής,

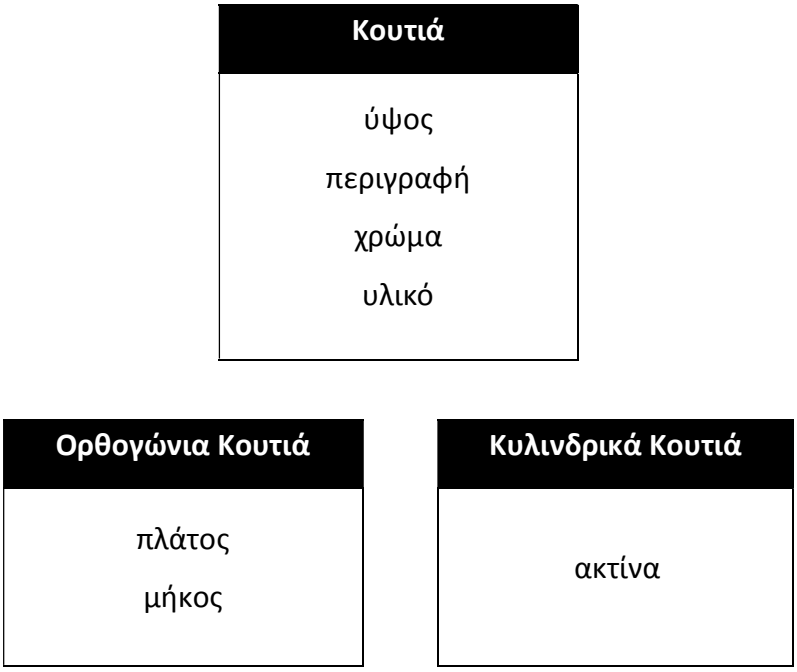
Ορθογώνια Κουτιά
πλάτος
μήκος
ύψος
περιγραφή
χρώμα
υλικό

Κυλινδρικά Κουτιά
ύψος,
ακτίνα
περιγραφή
χρώμα
υλικό

Από την αντιπαράθεση των δυο αυτών κλάσεων παρατηρούμε ότι, υπάρχουν στοιχεία που επαναλαμβάνονται και στοιχεία που δεν επαναλαμβάνονται. Αυτό θα μπορούσε να σημαίνει

ενδεχομένως ότι στην συνέχεια της ανάπτυξης του λογισμικού, θα είμαστε αναγκασμένοι να δημιουργούμε μεθόδους για όμοιες λειτουργικότητες. Για παράδειγμα, έναν έλεγχο ορθότητας, ώστε να μην δέχονται τα ΕΛ.ΤΑ. για φόρτωση, υπερμεγέθη κιβώτια σε ύψος, ή έναν έλεγχο για την αποτροπή αποδοχής κουτιού χωρίς διαστάσεις, ή χωρίς τύπο υλικού. Θα έπρεπε δηλαδή να κατασκευάσουμε μια μέθοδο για να υπολογίζει ένα ανώτατο όριο ύψος στα κουτιά και να βγάζει ένα ενημερωτικό μήνυμα αλλά και μια παρόμοια μέθοδο για να υπολογίζει επίσης το ανώτατο όριο ύψος στους κυλίνδρους. Και φυσικά αυτό θα μπορούσε να αφορά και πολλές άλλες μεθόδους ελέγχου, για το υλικό, τις ώρες παραλαβής, το ανώτατο βάρος, κ.λπ. Αυτή η περιγραφή των καταστάσεων ελέγχου αναδεικνύει το πρόβλημα της συγγραφής κώδικα που στην πραγματικότητα θα εκτελεί τα ίδια πράγματα. Μήπως λοιπόν για να το αποφύγουμε αυτό, θα ήταν καλύτερα να βγάλουμε τα κοινά στοιχεία, σε μια κοινή κλάση και τα ειδικά στοιχεία να τα αφήσουμε σε δυο υποκλάσεις για την κάθε περίπτωση κουτιού. Συνεπώς, οι δύο αρχικές κλάσεις ΟΡΘΟΓΩΝΙΑ ΚΟΥΤΙΑ και ΚΥΛΙΝΔΡΙΚΑ ΚΟΥΤΙΑ, μπορούν να διασπαστούν σε τρεις κλάσεις και στην νέα Ο.Ο.Ρ., σχεδίαση μας να αποτυπωθούν ως, ΚΟΥΤΙΑ, ΟΡΘΟΓΩΝΙΑ ΚΟΥΤΙΑ και ΚΥΛΙΝΔΡΙΚΑ ΚΟΥΤΙΑ.

Αυτή ακριβώς είναι η ιδέα, τα στοιχεία που είναι κοινά να τοποθετούνται σε μια υπέρ-κλάση (superclass) και τα διαφορετικά στοιχεία να παραμένουν σε υπό-κλάσεις (subclass), απογόνους (ή κληρονόμους) που κληρονομούν τα κοινά χαρακτηριστικά, που βρίσκονται από πάνω τους.



Αυτό στην αντικειμενοστρέφεια ονομάζεται κληρονομικότητα και αποτελεί ίσως την σπουδαιότερη τεχνική της ευκολία. Η κλάση ΚΟΥΤΙΑ, είναι μια υπέρ-κλάση και οι κλάσεις ΟΡΘΟΓΩΝΙΑ ΚΟΥΤΙΑ και ΚΥΛΙΝΔΡΙΚΑ ΚΟΥΤΙΑ είναι δυο άλλες υπό-κλάσεις οι οποίες κληρονομούν τα κοινά χαρακτηριστικά της υπέρ-κλάσης. Συνεπώς ένα καλογραμμένο πρόγραμμα διασπάται από τον κατασκευαστή κατά το στάδιο της σχεδίασης, σε επιμέρους κλάσεις με στόχο να απλοποιείται η ανάπτυξη του στην συνέχεια και να κατασκευάζονται ρουτίνες, συναρτήσεις, μέθοδοι μια φορά χωρίς άχρηστες επαναλήψεις. Η κληρονομικότητα στην αντικειμενοστρέφεια εμφανίζεται σε απλή μορφή αλλά και σε πολλαπλή μορφή. Την κληρονομικότητα μπορούμε να την συναντήσουμε και στον ακολουθιακό προγραμματισμό και συνεπώς δεν θα πρέπει να θεωρείται αποκλειστικότητα του Ο.Ο.Ρ.

5.6. Υπό-Κλάσεις – subclass

Η έννοια της *υπό-κλάσης* (subclass), περιγράφηκε στις προηγούμενες ενότητες και είναι μια κατάταξη αντικειμένων τα οποία δανείζονται ή κληρονομούν περιεχόμενο από μια ανώτερη κλάση. Η γνώση της έννοιας, θα μπορούσε να εμπλουτιστεί και με το ότι, οι κλάσεις δεν είναι απαραίτητο να φιλοξενούν μόνο δεδομένα (ύψος, περιγραφή, κ.λπ.) αλλά θα μπορούσαν να αφορούν καθαρό κώδικα. Η σχέση δηλαδή της γενίκευσης – εξειδίκευσης, μπορεί να αφορά

και μια υπό-ρουτίνα η οποία κληρονομεί χαρακτηριστικά λειτουργικότητας από μια ανώτερη ρουτίνα.

5.7. Ενθυλάκωση – encapsulation

Με τον όρο *ενθυλάκωση* (encapsulation), στην αντικειμενοστρέφεια περιγράφουμε μια κατάσταση αφαιρετικότητας, η οποία έχει σκοπό να αποκρύπτει λεπτομέρειες από το να είναι ορατές, έτσι ώστε να απλοποιεί την παρακολούθηση ενός συστατικού λογισμικού. Κάποια δηλαδή από τα στοιχεία της κλάσης βρίσκονται νοητά καλυμμένα, ενθυλακωμένα, από μια περιβάλλουσα «μεμβράνη» απόκρυψης. Η τεχνική στην ελληνική βιβλιογραφία παρουσιάζεται και με τον όρο *κελυφοποίηση*, του κώδικα σε επίπεδα θέασης. Η κλάση δηλαδή αποκρύπτει τα στοιχεία της και δεν χρειάζεται να είναι ορατή και η παραμικρή της λεπτομέρεια. Αρκεί το συστατικό λογισμικό να είναι λειτουργικό από μια κατάλληλη διεπαφή και να επιτελεί σωστά μια διεργασία. Για παράδειγμα, εάν στην κλάση ΜΑΘΗΜΑΤΑ, θέλαμε να κατασκευάσουμε μια μέθοδο την οποία όταν θα την καλούμε θα αλλάζει την ώρα του μαθήματος κατά +1 ώρα, η κλήση και μόνο της ρουτίνας με όρισμα εισόδου, το όνομα του μαθήματος είναι αρκετό για να αλλάξει την ώρα, χωρίς να μας απασχολήσει πως είναι φτιαγμένη η κλάση ΜΑΘΗΜΑΤΑ, τι μεταβλητές περιλαμβάνει και πως η μέθοδος υπολογίζει και αλλάζει την ώρα.

5.8. Αφαιρετικότητα – abstraction

Η *αφαιρετικότητα* (abstraction) αφορά την προσέγγιση μιας κλάσης μέσω μιας συνάρτησης η οποία χρησιμοποιεί μέρος των χαρακτηριστικών της κλάσης και αγνοεί όλα τα υπόλοιπα στοιχεία της. Για παράδειγμα μια μέθοδος της κλάσης ΚΟΥΤΙΑ, μπορεί να αναζητά το υλικό του κουτιού, αγνοώντας παντελώς την ύπαρξη όλων των άλλων χαρακτηριστικών της κλάσης (περιγραφή, χρώμα, ύψος, κ.λπ.).

5.9. Υπερφόρτωση – overloading

Με τον όρο *υπερφόρτωση* (overloading), στην αντικειμενοστρέφεια περιγράφεται η δυνατότητα να δημιουργούνται ρουτίνες (μέθοδοι) οι οποίες ενώ έχουν το ίδιο όνομα, περιέχουν διαφορετικό κώδικα και παράγουν διαφορετικά λειτουργικά αποτελέσματα. Αυτό επιτυγχάνεται με την διαφοροποίηση στα ορίσματα εισόδου. Μια ρουτίνα δηλαδή,

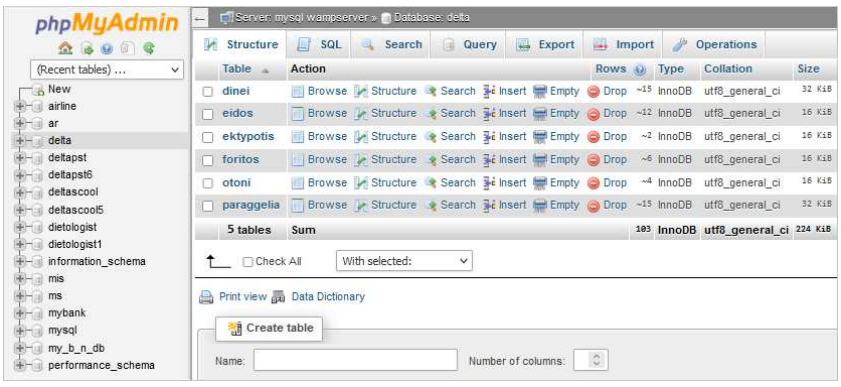
αναπτύσσεται με το ίδιο όνομα πολλές φορές αλλά με διαφορετικό σώμα εντολών και σε κάθε μια αυτές τις εκδόσεις τα ορίσματα εισόδου διαφέρουν. Η διαφοροποίηση μπορεί να αφορά το πλήθος αλλά και τους τύπους των ορισμάτων. Αυτή η διαφοροποίηση, στα ορίσματα εισόδου, επιτρέπει στην γλώσσα να αναγνωρίζει την κατάλληλη έκδοση από τις συνώνυμες μεθόδους που θα πρέπει να ενεργοποιήσει.

5.10. Πολυμορφισμός – polymorphism

Με τον όρο *πολυμορφισμός* (polymorphism), η αντικειμενοστρέφεια περιγράφει την δυνατότητα, μεταξύ κλάσεων οι οποίες βασίζονται στην σχέση γενίκευσης – εξειδίκευσης, να δημιουργούνται ρουτίνες (μέθοδοι), οι οποίες να έχουν το ίδιο όνομα, να περιέχουν διαφορετικό κώδικα, να παράγουν διαφορετικά λειτουργικά αποτελέσματα, επενεργώντας επάνω σε διαφορετικά αντικείμενα. Αυτό η γλώσσα το επιτυγχάνει βασιζόμενη στον τύπο του τρέχοντος αντικειμένου για το οποίο καλείται η συνώνυμη μέθοδος. Εάν δηλαδή το αντικείμενο είναι συμβατό με την υπέρ-κλάση καλείται η μέθοδος από εκεί, ενώ εάν το αντικείμενο είναι συμβατό με την υπό-κλάση καλείται η μέθοδος της υποκλάσης και ας έχουν το ίδιο όνομα. Η συνάρτηση δηλαδή καταλαβαίνει τι πρέπει να κάνει, βλέποντας τον τύπο του αντικειμένου για το οποίο καλείται και από εκεί και η εξήγηση του όρου πολύ-μορφισμός.

5.11. Άλλες Τεχνοτροπίες Προγραμματισμού

Εκτός από τις παραδοσιακές τεχνοτροπίες της διαδικασιακής προσέγγισης και της αντικειμενοστρέφειας που ήδη μελετήσαμε, η τεχνολογία λογισμικού προσφέρει και άλλες διαφορετικές διαδικασίες για την ανάπτυξη συστημάτων και την κατασκευή εφαρμογών λογισμικού. Για παράδειγμα, μια κυρίαρχη κατάσταση στην ανάπτυξη των σύνθετων συστημάτων λογισμικών είναι οι βάσεις δεδομένων (Data Bases) και η ενσωματωμένη γλώσσα που περιλαμβάνουν και ονομάζεται S.Q.L. (Structured Query Language). Οι βάσεις δεδομένων είναι μια ανώτερη οργανωμένη δομή δεδομένων και βασίζεται επάνω σε κατάλληλους και πολύπλοκους εννοιολογικούς συσχετισμούς σε σύνθετες συλλογές δομές δεδομένων. Οι βάσεις δεδομένων προσεγγίζονται μέσα από κατάλληλα υποβοηθητικά γραφικά περιβάλλοντα εργασίας που ονομάζονται, *συστήματα διαχείρισης βάσεων δεδομένων* (Data Base Managenent Systems). Η S.Q.L. είναι μια εξαιρετικά ισχυρή γλώσσα δημιουργίας ερωτημάτων προς μια βάση δεδομένων και προσφέρει στον χειριστή προγραμματιστή έναν αυτοματοποιημένο τρόπο χειρισμού των δεδομένων, χωρίς την ανάγκη συγγραφής ενός διαδικασιακού ή αντικειμενοστραφούς προγράμματος. Με την S.Q.L. ο προγραμματιστής δεν χρειάζεται να δημιουργήσει κάποιο πρόγραμμα, απλά περιγράφει στον υπολογιστή, τα δεδομένα που θέλει και ο υπολογιστής αναλαμβάνει να τα εξάγει για λογαριασμό του και να τα παρουσιάσει. Αυτός είναι και ο βασικός λόγος της απόλυτης επικράτησης των βάσεων δεδομένων σε όλες σχεδόν τις σύγχρονες ψηφιακές εφαρμογές, από την πλοήγηση μας στο internet, τις δοσοληψίες με τα κινητά, τα ραντεβού με τον ιατρό μας, τον τραπεζικό μας λογαριασμό, το αεροπορικό μας ταξίδι αλλά και κάθε άλλη ψηφιακή μας δραστηριότητα. Πίσω από αυτό το σύγχρονο ψηφιακό μας κόσμο υπάρχει η γλώσσα S.Q.L. (Structured Query Language).



Σύνοψη κεφαλαίου

Στο πέμπτο κεφάλαιο παρουσιάστηκαν σημαντικές έννοιες από την διαδικασία της ανάπτυξης προγραμμάτων με την μεθοδολογία της αντικειμενοστρέφειας. Η μελέτη περιελάμβανε την χρήση καταστάσεων και παραδειγμάτων από την καθημερινή ζωή των εκπαιδευομένων και μέσω αυτών επεξηγήθηκαν δύσκολοι όροι της αντικειμενοστρέφειας.

Ερωτήσεις αξιολόγησης

Ερώτηση-1: Περιγράψτε τι είναι ένα αντικείμενο μιας κλάσης;

Ερώτηση-2: Περιγράψτε τι είναι η κλάση;

Ερώτηση-3: Περιγράψτε τι είναι η κληρονομικότητα στην αντικειμενοστρέφεια;

Ερώτηση-4: Είναι σωστό, ότι μια υπό-κλάση κληρονομεί από μια υπέρ-κλάση;

Ερώτηση-5: Είναι σωστό, ότι μια υπέρ-κλάση κληρονομεί από μια υπό-κλάση;

Ερώτηση-6: Είναι σωστό ότι, μια κλάση έχει στο εσωτερικό της μεθόδους;

Ερώτηση-7: Είναι σωστό ότι, μια κλάση μπορεί να έχει στο εσωτερικό μόνο μεθόδους;

Ερώτηση-8: Είναι σωστό ότι, στιγμιότυπο, είναι ένα αντικείμενο μιας κλάσης σε μια συγκεκριμένη χρονική στιγμή;

Ερώτηση-9: Είναι σωστό ότι, μέθοδος, είναι μια συνάρτηση και βρίσκεται μέσα σε μια κλάση;

Ερώτηση-10: Είναι σωστό ότι, μέθοδος, είναι μια ρουτίνα και βρίσκεται μέσα σε μια κλάση;

Ερώτηση-11: Είναι σωστό ότι, συνάρτηση, είναι μια μέθοδος και βρίσκεται μέσα σε μια κλάση;

Απαντήσεις ερωτήσεων αξιολόγησης

Απάντηση-1: Συμβουλευτείτε τις εισαγωγικές παρατηρήσεις και την ενότητα 5.1. του κεφαλαίου.

Απάντηση-2: Συμβουλευτείτε την ενότητα 5.2. του κεφαλαίου.

Απάντηση-3: Συμβουλευτείτε την ενότητα 5.5. του κεφαλαίου.

Απάντηση-4: Ναι, είναι σωστό.

Απάντηση-5: Όχι, είναι λάθος.

Απάντηση-6: Ναι, είναι σωστό.

Απάντηση-7: Ναι, είναι σωστό.

Απάντηση-8: Ναι, είναι σωστό.

Απάντηση-9: Ναι, είναι σωστό.

Απάντηση-10: Ναι, είναι σωστό.

Απάντηση-11: Ναι, είναι σωστό.

6. ΕΡΓΑΣΤΗΡΙΑΚΗ ΜΕΛΕΤΗ

Σκοπός και επιμέρους στόχοι

Ο σκοπός του τελευταίου κεφαλαίου του συγγράμματος είναι η εργαστηριακή μελέτη, της γλώσσας προγραμματισμού C++, μέσω της μετατροπής ενός σύνθετου αλγορίθμου σε κατάλληλες εντολές προγραμματισμού. Οι εκπαιδευόμενοι καλούνται με βάση μια συγκεκριμένη κατασκευή ενός λογισμικού έκδοσης κοινοχρήστων κατοικιών, να μετατρέψουν τις εμπειρίες από την μελέτη τους στις γλώσσες προγραμματισμού και την C++, σε ένα πλήρες λειτουργικό πρόγραμμα.

Προσδοκώμενα αποτελέσματα

Με την συμμετοχή τους σε αυτή την εργαστηριακή διαδικασία οι εκπαιδευόμενοι θα,

- εφαρμόσουν στην πράξη όλες τις προγραμματιστικές τεχνικές, τις οποίες διδάχτηκαν για την γλώσσα C++, μέσα από τα συγγράμματα της σειράς, Προγραμματισμός Ηλεκτρονικών Υπολογιστών & Μηχανών, της ΑCTA,
- προσπαθήσουν να γράψουν ένα σύνθετο λειτουργικό πρόγραμμα,
- διδαχτούν από τα λάθη τους,
- συγκρίνουν την κατασκευή τους με μια ολοκληρωμένη επίλυση C++, με υποβοηθητικούς σχολιασμούς, δομημένη συγγραφή και κανόνες αμυντικού προγραμματισμού.

6.1. Εκφώνηση εργασίας

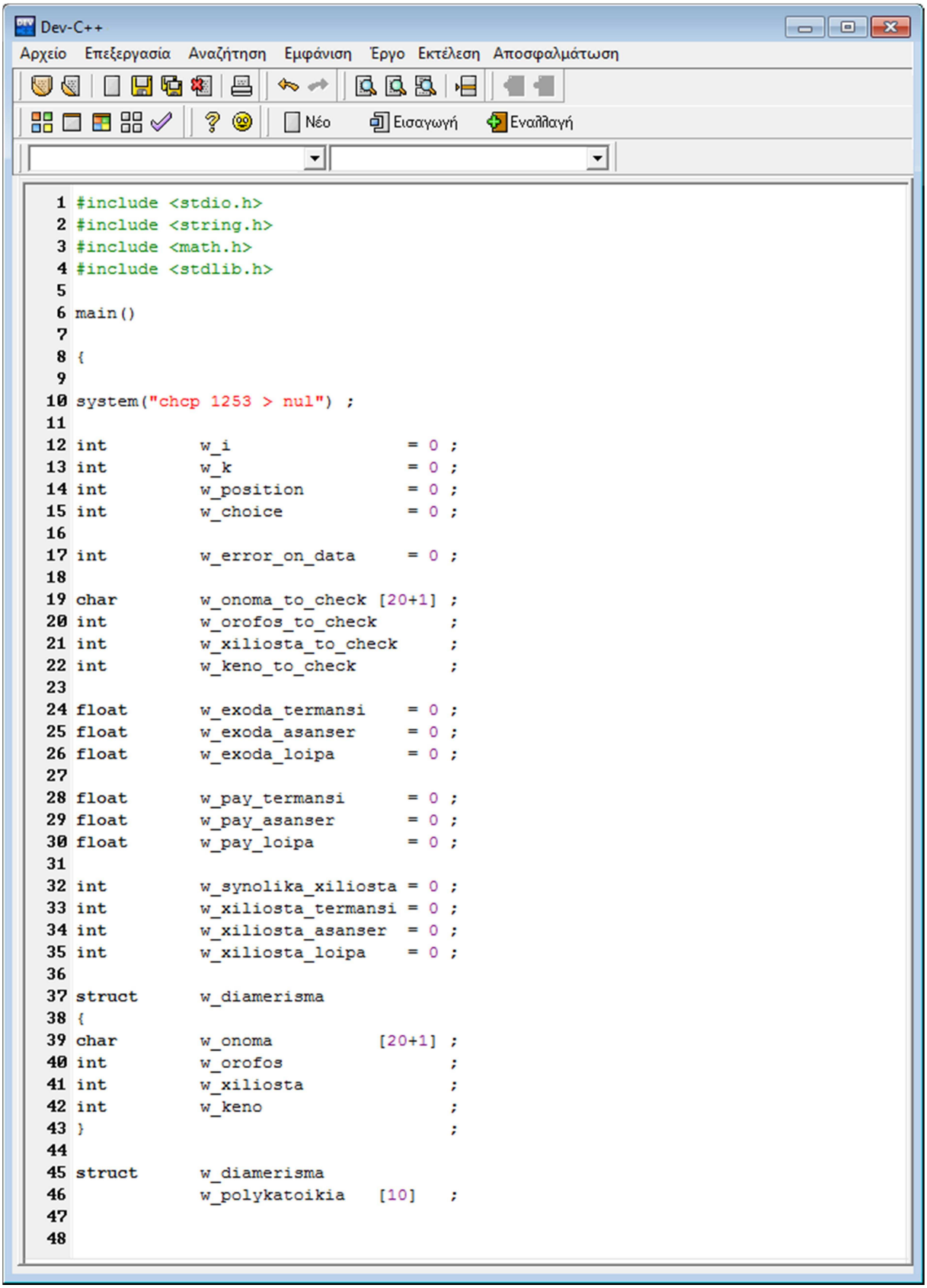
Κατασκευάστε ένα πρόγραμμα σε C++ με την χρήση του compiler Dev-C++, το οποίο να υποστηρίζει την έκδοση κοινοχρήστων των 10 διαμερισμάτων, της πολυκατοικίας σας. Ειδικότερα, το πρόγραμμα σας θα πρέπει να διαθέτει κατάλληλη διεπαφή μέσα από την οποία θα εισάγονται στο πρόγραμμα και για κάθε διαμέρισμα, επώνυμο χρήστη, όροφος διαμερίσματος, χιλιοστά διαμερίσματος και μια ένδειξη (διακόπτης) εάν το διαμέρισμα κατοικείται ή είναι κενό. Ακολούθως, μετά από την εισαγωγή των στοιχείων των διαμερισμάτων, η διεπαφή να ζητά, τα συνολικά έξοδα θέρμανσης, τα έξοδα συντήρησης για το ασανσέρ και ένα συνολικό ποσό για τα υπόλοιπα ποσά. Το πρόγραμμα στην συνέχεια θα πρέπει, να κάνει επιμερισμό των εξόδων, ως εξής, α) οι κατοικίες του ισογείου δεν επιβαρύνονται από το ασανσέρ, τα κλειστά διαμερίσματα δεν θα επιβαρύνονται από την θέρμανση, β) όταν ένα διαμέρισμα δεν συμμετέχει σε κάποιο έξοδο, η συμμετοχή των υπόλοιπων διαμερισμάτων θα πρέπει να κατανέμεται σε σχέση με το άθροισμα των χιλιοστών των διαμερισμάτων που συμμετέχουν και γ) όλα τα άλλα έξοδα θα κατανέμονται, με βάση τα χιλιοστά τους.

ΕΝΟΙΚΟΣ	ΟΡΟΦΟΣ	ΧΙΛΙΟΣΤΑ	ΚΕΝΟ	ΘΕΡΜΑΝΣΗ	ΑΣΑΝΣΕΡ	ΛΟΙΠΑ	ΣΥΝΟΛΟ
Παπαδάκης	0ος	100		200,30	0,0	20,34	220,64
Μελέτης	1ος	130		100,00	30,0	40,00	170,00

Το πρόγραμμα μετά από την ολοκλήρωση της λειτουργίας της διεπαφής, θα πρέπει να τυπώνει στην οθόνη μια συγκεντρωτική εκτύπωση με αναλυτικά στοιχεία για όλα τα διαμερίσματα όπως ακριβώς εμφανίζεται στην εικόνα της εκφώνησης. Η ανάπτυξη του προγράμματος θα πρέπει να περιλαμβάνει σύνθετες δομές (πίνακες και κατασκευές νέων τύπων δεδομένων με την εντολή **struct**) όπως αυτές περιγράφηκαν και επεξηγήθηκαν στα αντίστοιχα κεφάλαια των συγγραμμάτων της σειράς.

Σημειώσεις: α) για κλειστό διαμέρισμα το αντίστοιχο πεδίο θα έχει την τιμή 1, β) τα συνολικά χιλιοστά (ανά έξοδο) δεν πρέπει να ξεπερνούν τον αριθμό 1000 και γ) οι όροφοι να αρχίζουν από το ισόγειο με τον αριθμό κατοικίας 0.

6.2. Ενδεικτική επίλυση




```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

main()

{

system("chcp 1253 > nul") ;

int         w_i                 = 0 ;
int         w_k                 = 0 ;
int         w_position          = 0 ;
int         w_choice            = 0 ;

int         w_error_on_data     = 0 ;

char        w_onoma_to_check [20+1] ;
int         w_orofos_to_check       ;
int         w_xiliosta_to_check     ;
int         w_keno_to_check         ;

float       w_exoda_termansi    = 0 ;
float       w_exoda_asanser     = 0 ;
float       w_exoda_loipa       = 0 ;

float       w_pay_termansi      = 0 ;
float       w_pay_asanser       = 0 ;
float       w_pay_loipa         = 0 ;

int         w_synolika_xiliosta = 0 ;
int         w_xiliosta_termansi = 0 ;
int         w_xiliosta_asanser  = 0 ;
int         w_xiliosta_loipa    = 0 ;

struct      w_diamerisma
{
char        w_onoma          [20+1] ;
int         w_orofos                ;
int         w_xiliosta              ;
int         w_keno                  ;
}                                   ;

struct      w_diamerisma
            w_polykatoikia   [10]   ;
```

Dev-C++

```
49 for ( w_i=0; w_i<10; w_i++ )            /* KANOYME INIT THN DOMH */
50 {
51      for ( w_k=0; w_k<21; w_k++ )
52      {
53          w_polykatoikia[w_i].w_onoma[w_k] = ' ' ;
54      }
55      w_polykatoikia[w_i].w_orofos   = 0 ;
56      w_polykatoikia[w_i].w_xiliosta = 0 ;
57      w_polykatoikia[w_i].w_keno     = 0 ;
58 }
59
60 do
61 {
62
63 printf("\nΕ Π Ι Λ Ο Γ Ε Σ   Π Ρ Ο Γ Ρ Α Μ Μ Α Τ Ο Σ\n\n") ;
64 printf("1: ΝΕΑ ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΩΝ ΔΙΑΜΕΡΙΣΜΑΤΟΣ\n") ;
65 printf("2: ΚΑΤΑΧΩΡΗΣΗ ΕΞΟΔΩΝ\n") ;
66 printf("3: ΥΠΟΛΟΓΙΣΜΟΣ & ΚΑΤΑΣΤΑΣΗ ΑΝΑΛΟΓΙΑΣ ΕΞΟΔΩΝ\n") ;
67 printf("4: ΕΞΟΔΟΣ\n") ;
68 printf("\n--> ΠΑΡΑΚΑΛΩ, ΔΩΣΤΕ ΕΠΙΛΟΓΗ... ") ;
69 scanf("%d", &w_choice) ;
70
71 switch (w_choice)
72 {
73
74     case 1:
75
76         w_position = -1 ;
77
78         for ( w_i=0; w_i<10; w_i++ )     /* DIATREXOYME GIA ELEYTERH TESH */
79         {
80             {
81             if ( w_polykatoikia[w_i].w_onoma[1] == ' ' )
82                 {
83                 w_position = w_i ;
84                 w_i        = 99  ;       /* GIA NA FEYGEI APO TO LOOP */
85                 }
86             }
87         }
88
89         {
90         if ( w_position == -1 )
91             {
92             printf("\n-----> ΟΛΑ ΤΑ ΔΙΑΜΕΡΙΣΜΑΤΑ ΕΧΟΥΝ ΗΔΗ ΔΗΛΩΘΕΙ\n") ;
93             }
94         else
95             {
96
97             w_orofos_to_check   = 0 ;
98             w_xiliosta_to_check = 0 ;
99             w_keno_to_check     = 0 ;
100            w_error_on_data     = 0 ;
101
102            printf("\n--> ΔΩΣΤΕ ΤΟ ΟΝΟΜΑ ΤΟΥ ΕΝΟΙΚΟΥ... ") ;
103            scanf("%s", w_onoma_to_check) ;
104
105            printf("\n--> ΔΩΣΤΕ ΤΟΝ ΟΡΟΦΟ (0-9)... ") ;
```

Dev-C++

```
106             scanf("%d", &w_orofos_to_check) ;
107
108             printf("\n--> ΔΩΣΤΕ ΤΑ ΧΙΛΙΟΣΤΑ (1-1000)... ") ;
109             scanf("%d", &w_xiliosta_to_check) ;
110
111             printf("\n--> ΕΙΝΑΙ ΚΕΝΟ=1 'Η /ΚΑΤΟΙΚΗΜΕΝΟ=0 ; ... ") ;
112             scanf("%d", &w_keno_to_check) ;
113
114             {
115             if ( w_orofos_to_check > 9 )
116                {
117                printf("\n-----> ΔΗΛΩΣΑΤΕ ΛΑΘΟΣ ΟΡΟΦΟ\n") ;
118                w_error_on_data = 1 ;
119                }
120             }
121
122             {
123             if ( w_xiliosta_to_check == 0 )
124                {
125                printf("\n-----> ΔΗΛΩΣΑΤΕ ΜΗΔΕΝΙΚΑ ΧΙΛΙΟΣΤΑ\n") ;
126                w_error_on_data = 1 ;
127                }
128             }
129
130             {
131             if ( w_xiliosta_to_check > 1000 )
132                {
133                printf("\n-----> ΔΗΛΩΣΑΤΕ ΥΠΕΡΒΟΛΙΚΑ ΧΙΛΙΟΣΤΑ\n") ;
134                w_error_on_data = 1 ;
135                }
136             }
137
138             {
139             if ( ( w_synolika_xiliosta + w_xiliosta_to_check ) > 1000 )
140                {
141                printf("\n-----> ΥΠΕΡΒΑΣΗ ΣΤΑ ΣΥΝ.ΧΙΛΙΟΣΤΑ ΤΟΥ ΚΤΙΡΙΟΥ\n") ;
142                w_error_on_data = 1 ;
143                }
144             }
145
146             {
147             if ( w_keno_to_check > 1 )
148                {
149                printf("\n-----> ΔΗΛΩΣΑΤΕ ΛΑΘΟΣ ΚΑΤΑΣΤΑΣΗ ΔΙΑΜΕΡΙΣΜΑΤΟΣ\n") ;
150                w_error_on_data = 1 ;
151                }
152             }
153
154             {
155             if ( w_error_on_data == 0 )
156                 {
157
158                 for ( w_i=0; w_i<20; w_i++ )
159                /* AGNWOYME TOYS > 20 XARAKTHRES
160                   GIA NA XWRESOYN STHN OTONH MAS */
161                 {
```

Dev-C++

```
162                     w_polykatoikia[w_position].w_onoma[w_i] =
163                                     w_onoma_to_check[w_i] ;
164                 }
165
166                 w_polykatoikia[w_position].w_orofos    = w_orofos_to_check ;
167                 w_polykatoikia[w_position].w_xiliosta = w_xiliosta_to_check ;
168                 w_polykatoikia[w_position].w_keno      = w_keno_to_check ;
169
170                 w_synolika_xiliosta = w_synolika_xiliosta +
171                                     w_xiliosta_to_check ;
172                 }
173             }
174
175             }
176
177         }
178
179         printf("\n\n\n\n") ;
180
181         break ;
182
183     case 2:
184
185         w_exoda_termansi = 0 ;
186         w_exoda_asanser  = 0 ;
187         w_exoda_loipa    = 0 ;
188
189         printf("\n--> ΔΩΣΤΕ ΤΑ ΣΥΝΟΛΙΚΑ ΕΞΟΔΑ ΘΕΡΜΑΝΣΗΣ... ") ;
190         scanf("%f", &w_exoda_termansi) ;
191
192         printf("\n--> ΔΩΣΤΕ ΤΑ ΣΥΝΟΛΙΚΑ ΕΞΟΔΑ ΓΙΑ ΤΟ ΑΣΑΝΣΕΡ... ") ;
193         scanf("%f", &w_exoda_asanser) ;
194
195         printf("\n--> ΔΩΣΤΕ ΤΟ ΣΥΝΟΛΟ ΤΩΝ ΛΟΙΠΩΝ ΕΞΟΔΩΝ... ") ;
196         scanf("%f", &w_exoda_loipa) ;
197
198                 /* DEN KANOYME SE KANENA EXODO, ELEGXO GIA ARNHTIKA POSA,
199                    EPEIDH MPOREI NA YPARXOYN AXODA APO PISTWTIKA TIMOLOGIA
200                    DHLADH ME ARNHTIKA POSA */
201
202         printf("\n\n\n\n") ;
203
204         break ;
205
206     case 3:
207
208         w_xiliosta_termansi = 0 ;
209         w_xiliosta_asanser  = 0 ;
210         w_xiliosta_loipa    = 0 ;
211                 /* TA LOIPA EXODA PANE SE OLA TA DIAMERISMATA, ALLA
212                    EPILEGOYME NA ELENXOYME KAI TA XILIOSTA GIA TA
213                    LOIPA EXODA GIA NA KALOIPSOYME KAI THN PERIPTWSH
214                    NA MHN DHLWTOYN APO TON XRHSTH OLA TA DIAMERISMATA */
215
216         for ( w_i=0; w_i<10; w_i++ )     /* DIATREXOYME TA DIAMERISMATA */
217         {
218             {
```

Dev-C++

```
219             if ( w_polykatoikia[w_i].w_onoma[1] != ' ' )
220                 {
221                /* STHN SYNEXEIA ARXIZOYME NA FTIAXNOYME TA XILIOSTA */
222                 {
223                 if ( w_polykatoikia[w_i].w_keno == 0 )
224                     {
225                     w_xiliosta_termansi = w_xiliosta_termansi +
226                                           w_polykatoikia[w_i].w_xiliosta ;
227                     }
228                 }
229
230                 {
231                 if ( w_polykatoikia[w_i].w_orofos > 0 )
232                     {
233                     w_xiliosta_asanser = w_xiliosta_asanser +
234                                          w_polykatoikia[w_i].w_xiliosta ;
235                     }
236                 }
237
238                 w_xiliosta_loipa = w_xiliosta_loipa +
239                                    w_polykatoikia[w_i].w_xiliosta ;
240
241                 }
242             }
243         }
244
245         printf("\n\n\n\n") ;
246         printf("-------------------------------------------------------------------
------------------------\n") ;
247         printf("                  K A T A S T A S H     A N A L O G I A S     E X
O D W N                \n") ;
248         printf("-------------------------------------------------------------------
------------------------\n") ;
249         printf("  EPWNYMIA              OROFOS XILIOSTA KENO  QERMANSH   ASANSEP
    YPOLOIPA   SYNOLO   \n") ;
250         printf("-------------------------------------------------------------------
------------------------\n") ;
251
252         for ( w_i=0; w_i<10; w_i++ )
253                /* DIATREXOYME TA DIAMERISMATA GIA NA TYPWSOYME */
254         {
255             {
256             if ( w_polykatoikia[w_i].w_onoma[1] != ' ' )
257                 {
258                 w_pay_termansi = 0 ;
259                 w_pay_asanser  = 0 ;
260                 w_pay_loipa    = 0 ;
261                /* STHN SYNEXEIA BRISKOYME TA POSA GIA KA8E DIAMERISMA */
262
263                 {
264                 if ( ( w_exoda_termansi != 0 ) && ( w_xiliosta_termansi != 0 ) )
265                     {
266                     w_pay_termansi = ( w_exoda_termansi *
267                                        w_polykatoikia[w_i].w_xiliosta ) /
268                                        w_xiliosta_termansi ;
269                     }
270                 }
```

Dev-C++

```
271                 {
272                 if ( ( w_exoda_asanser != 0 ) && ( w_xiliosta_asanser != 0 ) )
273                     {
274                     w_pay_asanser  = ( w_exoda_asanser *
275                                        w_polykatoikia[w_i].w_xiliosta ) /
276                                        w_xiliosta_asanser ;
277                     }
278                 }
279                 {
280                 if ( ( w_exoda_loipa != 0 ) && ( w_xiliosta_loipa != 0 ) )
281                     {
282                     w_pay_loipa    = ( w_exoda_loipa    *
283                                        w_polykatoikia[w_i].w_xiliosta ) /
284                                        w_xiliosta_loipa ;
285                     }
286                 }
287
288                 printf("%-20s\t%d      %03d      %d     %-7.1f    %-7.1f    %-7.
    1f    %-7.1f\n",
289                 w_polykatoikia[w_i].w_onoma,
290                 w_polykatoikia[w_i].w_orofos,
291                 w_polykatoikia[w_i].w_xiliosta,
292                 w_polykatoikia[w_i].w_keno,
293                 w_pay_termansi,
294                 w_pay_asanser,
295                 w_pay_loipa,
296                 w_pay_termansi + w_pay_asanser + w_pay_loipa
297
298                 ) ;
299
300                 }
301             }
302         }
303
304         printf("\n\n\n\n") ;
305
306         break ;
307
308 }
309
310 } while ( w_choice != 4 ) ;
311
312 system("PAUSE") ;
313
314 };
315
```

Γλωσσάριο σημαντικών όρων

Structured Query Language

Unified Modeling Language

Αλφαριθμητική μεταβλητή

Αμυντικός προγραμματισμός

Ανάλυση

Αναδιάταξη πίνακα

Ανακύκλωση

Αναφορά

Αντικείμενο

Αντικειμενοστραφής σχεδίαση

Αντικειμενοστραφής τεχνολογία προγραμματισμού

Αποτελεσματικότητα

Απόφαση

Αριθμητικοί τελεστές

Αρχεία

Αρχικοποίηση μεταβλητής

Αφαιρετικότητα

Βάσεις δεδομένων

Βιβλιοθήκες

Γράφοι

Γραμματική γλώσσας

Γραμμικές δομές δεδομένων

Δέντρα

Δείκτης μνήμης

Δείκτης πίνακα

Δεδομένα

Δεσμευμένες λέξεις

Διάγραμμα

Διάγραμμα ροής

Διάρκεια ζωής μεταβλητής

Διαγραφή στοιχείου

Διαδίκτυο

Διαδικασιακός προγραμματισμός

Διαπέραση πίνακα

Διατεταγμένος πίνακας

Διεπαφή

Διερμηνέας

Διευθυνσιοδότηση

Διεύθυνση μνήμης

Δομή

Δομή επανάληψης υπό συνθήκη

Δομή επιλογής

Δομημένες μορφές δεδομένων

Δομημένος προγραμματισμός

Δυναμικές δομές

Δυναμική διαχείριση μνήμης

Είσοδος αλγορίθμου

Εγγραφές

Εισαγωγή στοιχείου

Εκτελέσιμος κώδικας

Εμβέλεια μεταβλητής

Εμφωλιασμένες δομές επανάληψης

Ενημερωτικό μήνυμα

Ενθυλάκωση

Εντολές ανακατεύθυνσης

Εντολή

Εντολή do ... while ...

Εντολή for

Εντολή while

Έξοδος αλγορίθμου

Εξωτερική δομή επανάληψης

Επανάληψη με συνθήκη και βηματισμό

Επανάληψη με συνθήκη στην αρχή

Επανάληψη με συνθήκη στο τέλος

Επεξεργασία δεδομένων

Επιστρεφόμενη μεταβλητή

Επιστροφές συναρτήσεων

Εσωτερική δομή επανάληψης

Ηλεκτρονικός υπολογιστής Η/Υ

Θεμελιώδεις τύποι μεταβλητών

Κλάση

Κληρονομικότητα

Λίστες

Λογικοί τελεστές

Λογικό διάγραμμα

Λογισμικό

Μέγεθος μεταβλητής

Μέθοδος

Μενού επιλογών

Μερική διάρκεια ζωής μεταβλητής

Μετα-δεδομένα

Μεταβλητές

Μεταβλητή καθολικής εμβέλειας

Μεταβλητή τοπικής εμβέλειας

Μεταγλωττιστής

Μη γραμμικές δομές δεδομένων

Μη διατεταγμένος πίνακας

Ολοκληρωμένα περιβάλλοντα ανάπτυξης

Όνομα μεταβλητής

Ορίσματα εισόδου

Ορίσματα εξόδου

Οργανωτικό σχήμα

Πέρασμα ορίσματος με αναφορά (by reference)

Πέρασμα ορίσματος με τιμή (by value)

Πίνακες

Πίνακες πολλών διαστάσεων

Πεδίο

Περιγραφικά ονόματα μεταβλητών

Περιεχόμενο μεταβλητής

Περιοχή μνήμης

Πηγαίος κώδικας

Πλήρης διάρκεια ζωής μεταβλητής

Πληροφορίες

Πολυμορφισμός

Προγραμματισμός υπολογιστών

Προγραμματιστές

Προδιαγραφή

Προσομοίωση

Πρόγραμμα ηλεκτρονικού υπολογιστή

Πρότυπο C++20

Ροή εισόδου

Ροή εξόδου

Σταθερές

Σταθερή αναφορά

Στατικές δομές

Στηλοθέτηση κώδικα

Στιγμιότυπο

Στοιχειώδης μεταβλητή

Συγκριτικοί τελεστές

Συναρτήσεις

Σχεδίαση

Σχόλια

Σύνθετη μεταβλητή

Σύνταξη προγράμματος

Τακτοποίηση αντικειμένων

Τελεστής

Τελεστής έμμεσης διευθυνσιοδότησης

Τεχνικό εγχειρίδιο κατασκευαστή

Τεχνολογία λογισμικού

Τύπος μεταβλητής

Υπέρ-κλάση

Υπερφόρτωση

Υπό-κλάση

Ψευδοκώδικας

Aristotle Certification Training & Assessment
Τεχνοβλαστός Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης

Βιβλιογραφία

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ, Λουκάς Γεωργιάδης, Σταύρος Δ. Νικολόπουλος, Λεωνίδας Παληός, Σ.Ε.Α.Β., 2015.

Διαδικαστικός Προγραμματισμός (η γλώσσα C), Πάρις Μαστοροκώστας, Σ.Ε.Α.Β., 2015.

Στοιχεία Τεχνολογίας Λογισμικού, Βασίλειος Βεσκούκης, Σ.Ε.Α.Β., 2015.

Beginning C: From Novice to Professional, Ivor Horton, 4th. ed, APRESS, 2006.

C PROGRAMMING: A Modern Approach, K.N.King, 2nd. ed, W.W.NORTON, 2008.

OBJECT-ORIENTED ANALYSIS AND DESIGN With applications, Booch Grady, 2nd. ed, ADDISON-WESLEY, 1994.

Programming language C++, ISO/IEC JTC1 SC22 WG21 N4860, I.S.O./I.E.C., 2020.

The Art of Computer Programming, Donald Knuth, Addison-Wesley, 1968.

THE C PROGRAMMING LANGUAGE, Brian W. Kernighan, Dennis M. Ritchie, Prentice-Hall International, Bell Labs, 1978.

THE C++ PROGRAMMING LANGUAGE, Bjarne Stroustrup, 3rd. ed, AT & T Labs, Addison-Wesley, 1997.

The development of the C programming language, Dennis M. Ritchie, in History of Programming Languages JJ, Addison-Wesley, 1996.

Επίλογος - Οδηγίες για περαιτέρω μελέτη

Έχοντας φτάσει σε αυτό το σημείο, έχετε ολοκληρώσει την ανάγνωση των τριών συγγραμμάτων της σειράς, *Προγραμματισμός Ηλεκτρονικών Υπολογιστών & Μηχανών, της ACTA.* Η σειρά των συγγραμμάτων, δημιουργήθηκε με κατάλληλο σχεδιασμό προσβλέποντας σε συγκεκριμένους στόχους και μαθησιακά αποτελέσματα. Το εκπαιδευτικό υλικό που σταδιακά παρατέθηκε στους εκπαιδευόμενους, περιελάμβανε όλες εκείνες τις απαραίτητες προγραμματιστικές τεχνικές και τις βασικές μεθοδολογίες για να τους ωθήσει στην υιοθέτηση, της συμπεριφοράς του νέου προγραμματιστή και να τους προετοιμάσει για την συνέχεια. Η διαδικασία της εκπαίδευσης περιελάμβανε την εκτεταμένη χρήση της γλώσσας προγραμματισμού C++, ενός προγραμματιστικού περιβάλλοντος ικανού, να υποστηρίξει την δημιουργία οποιασδήποτε εφαρμογής, μικρής ή μεγάλης και για οποιαδήποτε πλατφόρμα, από ένα κινητό τηλέφωνο, έναν υπολογιστή, έως και μια διαδικτυακή εφαρμογή.

Ολοκληρώνοντας το τελικό αυτό στάδιο, οι αναγνώστες μπορούν να συνεχίσουν και να επεκτείνουν τις γνώσεις τους στον προγραμματισμό και την γλώσσα προγραμματισμού C, μέσα από την προτεινόμενη βιβλιογραφία.