ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΤΕ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΥΓΧΡΟΝΕΣ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

ΚΩΤΣΙΑΣ ΝΙΚΟΣ

ΤΣΙΑΠΑΛΗΣ ΣΩΤΗΡΗΣ

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ ΚΑΡΕΛΗΣ ΔΗΜΗΤΡΙΟΣ

ΑΡΙΘΜΟΣ ΠΤΥΧΙΑΚΗΣ: 1470

ΠΑΤΡΑ ΦΕΒΡΟΥΑΡΙΟΣ 2015

ΠΡΟΛΟΓΟΣ

Στην παρούσα πτυχιακή εργασία παρουσιάζουμε έναν οδηγό περιγραφής με τα βασικά χαρακτηριστικά και τις δυνατότητες τριών από τις πιο βασικές γλώσσες προγραμματισμού και συγκεκριμένα των C, C++ και Java.

Αρχικά παρουσιάζουμε τη γλώσσα προγραμματισμού C η οποία εφαρμόζει τις αρχές του διαδικασιακού ή δομημένου προγραμματισμού.Ο δομημένος προγραμματισμός βοηθάει την ευκολότερη συγγραφή πολύπλοκων προγραμμάτων. Με το δομημένο προγραμματισμό γίνεται πιο εύκολη η διαχείριση, η συντήρηση και η αποσφαλμάτωση σύνθετων προγραμμάτων, καθώς βασίζεται σε μεγαλύτερες, πολυπλοκότερες και περιεκτικότερες μονάδες που ονομάζονται συναρτήσεις ή διαδικασίες.

Ο δομημένος προγραμματισμός βασίζεται στην αρχή του διαίρει και βασίλευε, καθώς διασπά το βασικό πρόβλημα σε μικρότερα υπο-προβλήματα και υλοποιεί ξεχωριστά το κάθε υπο-πρόβλημα με μια συνάρτηση. Επαναλαμβανόμενες λειτουργίες μπορούν να υλοποιηθούν πολύ γρήγορα εκτελώντας τις ίδιες συναρτήσεις και όχι γράφοντας νέο κώδικα. Επίσης η δημιουργία βιβλιοθηκών με έτοιμο κώδικα μπορεί να διευκολύνει πολύ την ανάπτυξη νέων εφαρμογών καθώς οι βιβλιοθήκες αυτές μπορούν να χρησιμοποιηθούν σε οποιαδήποτε εφαρμογή δημιουργούμε.

Στη συνέχεια παρουσιάζουμε τη γλώσσα προγραμματισμού C++ η οποία αποτελεί επέκταση της C στο περιβάλλον του αντικειμενοστραφούς προγραμματισμού. Ο αντικειμενοστραφής προγραμματισμός στηρίζεται σε αρχές που έχουν τις ρίζες τους στη δεκαετία του '60 και συγκεκριμένα στη γλώσσα Simula. Η βασική φιλοσοφία του αντικειμενοστραφούς προγραμματισμού είναι να περιγράψει ένα σύστημα με φυσικό τρόπο. Για να επιτευχθεί αυτό, πρέπει να γίνει περιγραφή των αντικειμένων από τα οποία αποτελείται το σύστημα, να μελετηθεί η συμπεριφορά τους και ο τρόπος με τον οποίο αλληλεπιδρούν μεταξύ τους. Στην αντικειμενοστραφή προσέγγιση, το αντικείμενο είναι ένα προγραμματιστικό «πακέτο», το οποίο αποτελείται από διαδικασίες και δεδομένα σχετιζόμενα μεταξύ τους. Αυτές οι διαδικασίες ονομάζονται μέθοδοι ενώ τα δεδομένα ονομάζονται ιδιότητες. Οι μέθοδοι σε πολλές περιπτώσεις είναι γραμμένες σε κάποια διαδικαστική γλώσσα.Το πρόγραμμα, σε αυτήν την προσέγγιση, αποτελείται από αντικείμενα, τα οποία μπορούν να αλληλεπιδρούν μεταξύ τους με μηνύματα. Τα μηνύματα ενεργοποιούν τις μεθόδους των αντικειμένων, που με τη σειρά τους μπορεί να στείλουν άλλα μηνύματα.

Τέλος παρουσιάζουμε τη γλώσσα προγραμματισμού Java η οποία εστιάζει και αυτή στο περιβάλλον του αντικειμενοστραφούς προγραμματισμού όπως και η C++ και επιπλέον η Java επιτρέπει την κατασκευή εφαρμογών που «τρέχουν» στο διαδίκτυο και ονομάζονται μικρο-εφαρμογές (applets). Το βασικό της πλεονέκτημα της είναι ότι ένα πρόγραμμα γραμμένο σε Java μπορεί να εκτελεστεί σε οποιοδήποτε λειτουργικό σύστημα όπου βρίσκεται εγκατεστημένος ο διερμηνευτής που ονομάζεται Java Runtime. Ο διερμηνευτής αυτός μετατρέπει τις εντολές της Java σε εντολές του λειτουργικού συστήματος όπου είναι εγκατεστημένος και κάνει τη Java συμβατή με κάθε πλατφόρμα. Ένα applet της java είναι μια μικρή εφαρμογή που εκτελεί μια συγκεκριμένη λειτουργία και χρησιμοποιείται συχνά ως plug-in ενός μεγαλύτερου προγράμματος όπως π.χ. Flashmovieplayer. Τα Javaapplets παρέχουν εφαρμογές web με αλληλεπιδραστικά χαρακτηριστικά και σε συνδυασμό με την ανεξαρτησία της πλατφόρμας εκτέλεσης κατέστησαν την Java μια πολύ δημοφιλή γλώσσα προγραμματισμού.

ΠΕΡΙΛΗΨΗ

Η πτυχιακή εργασία είναι δομημένη σε 3 κεφάλαια. Σε κάθε κεφάλαιο παρουσιάζουμε μια γλώσσα προγραμματισμού αρχίζοντας από τη γλώσσα C. Πιο συγκεκριμένα πρώτα περιγράφουμε τα βασικά χαρακτηριστικά της C(μεταβλητές, σταθερές και τύποι δεδομένων). Στη συνέχεια αναφέρουμε τις εντολές εισόδου και εξόδου της Cκαι τις εντολές ελέγχου και επανάληψης παραθέτοντας κατάλληλα παραδείγματα σε κάθε εντολή. Ακολούθως περιγράφουμε τις πιο σύνθετες δομές δεδομένων της γλώσσας όπως π.χ. πίνακες (μονοδιάστατους και δισδιάστατους, δείκτες κ. λ. π.). Κάνουμε ιδιαίτερη αναφορά στα αλφαριθμητικά της Cκαι παραθέτουμε τις συναρτήσεις επεξεργασίας τους.Επίσης περιγράφουμε λεπτομερώς τις συναρτήσεις της γλώσσας, τον τρόπο κλήσης και επιστροφής αποτελεσμάτων κ.λ.π. Η τελευταία έννοια που παρουσιάζουμε είναι αυτή των δομών.

Στο 2^{ο} κεφάλαιο περιγράφουμε τη γλώσσα C++ ξεκινώντας από τις διαφορές που παρουσιάζει σε σχέση με τη γλώσσα C. Ακολούθως περιγράφουμε τα ιδιαίτερα χαρακτηριστικά των συναρτήσεων της C++ όπως για παράδειγμα τις εξορισμού παραμέτρους κα την υπερφόρτωση τους. Μετά αναφέρουμε ένα νέο χαρακτηριστικό της C++ που είναι οι αναφορές της και οι διαφορές της από τους δείκτες. Στη συνέχεια δίνουμε μεγάλο βάρος στις κλάσεις που χρησιμοποιεί η C++ και δίνουμε ιδιαίτερη έμφαση σε χαρακτηριστικά όπως δημιουργός και καταστροφέας. Μετά αναφέρουμε όλους τους τύπους κληρονομικότητας και δίνουμε αρκετά παραδείγματα πάνω σε αυτή. Το τελευταίο χαρακτηριστικό της γλώσσας είναι η υπερφόρτωση τελεστών (αριθμητικών και τελεστών εισόδου και εξόδου) καθώς και τα προβλήματα που αυτοί παρουσιάζουν.

Στο 3^{ο} κεφάλαιο παρουσιάζουμε τη γλώσσα Javaξεκινώντας από τα πλεονεκτήματα της Javaκαι τις διαφορές που παρουσιάζει σε σχέση με τη C++. Στη συνέχεια αναφέρουμε με αρκετή λεπτομέρεια τις κλάσεις στη Javaμε όλα τα χαρακτηριστικά τους και την κληρονομικότητα και τον πολυμορφισμό. Δίνουμε ιδιαίτερη έμφαση στις αφηρημένες κλάσεις και μεθόδους της Javaκαι στην έννοια των διεπαφών που είναι μια ειδική μορφή κλάσης και παραθέτουμε παραδείγματα με χρήση διεπαφών. Κατόπιν μελετούμε ένα ιδιαίτερο χαρακτηριστικό της javaπου είναι η διαχείριση εξαιρέσεων δηλ. η αντιμετώπιση σφαλμάτων εκτέλεσης με εκτύπωση διαγνωστικών μηνυμάτων. Επίσης μελετάμε τη διαχείριση αρχείων στη Java. Τέλος περιγράφουμε σύντομα τα appletsτης javaπου είναι εφαρμογές που τρέχουν στο διαδίκτυο και δείχνουμε πως μπορούμε να τα φορτώσουμε και να εκτελέσουμε.

ΠΕΡΙΕΧΟΜΕΝΑ

Α ΜΕΡΟΣ – ΓΛΩΣΣΑΣ

1 Διαδικασία συγγραφής προγραμμάτων σε γλώσσα C

Για να γράψουμε ένα πρόγραμμα σε C ακολουθούμε κατά σειρά τα ακόλουθα βήματα:

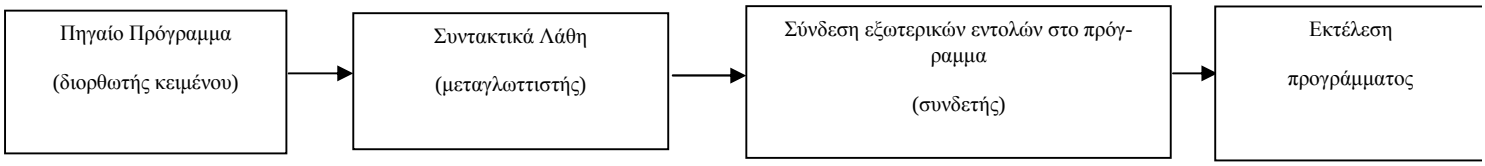
i. Γράφουμε πρώτα τις εντολές του προγράμματος μέσα στο διορθωτή κειμένου (editor). Ο διορθωτής κειμένου είναι ένας απλός κειμενογράφος με περιορισμένες δυνατότητες μορφοποίησης, εκτύπωσης κ.λ.π. Το πρόγραμμα που γράφουμε ονομάζεται πηγαίο (sourceprogram)

ii. Στη συνέχεια στέλνουμε το πηγαίο πρόγραμμα στο μεταγλωττιστή (compiler) ο οποίος αναλαμβάνει να μεταφράσει το πηγαίο πρόγραμμα σε γλώσσα μηχανής. Αυτό είναι εφικτό μόνο όταν ο μεταγλωττιστής δεν εντοπίσει στο πρόγραμμα μας συντακτικά λάθη. Αν υπάρχουν συντακτικά λάθη ο μεταγλωττιστής μας ενημερώνει για το ποια λάθη είναι αυτά και σε ποια γραμμή έγιναν. Συντακτικά είναι τα λάθη που οφείλονται στον τρόπο γραφής των προγραμμάτων π.χ. έλλειψη κάποιου ερωτηματικού, παρένθεσης κ.λ.π. Το πρόγραμμα που εξάγεται από τον μεταγλωττιστή ονομάζεται αντικείμενο (objectprogram).

iii. Στη συνέχεια συνδέουμε το αντικείμενο πρόγραμμα με τις βιβλιοθήκες της γλώσσας. Μια βιβλιοθήκη περιέχει εντολές κοινής χρήσης όπως π.χ. εντολές εισόδου-εξόδου, κοινές μαθηματικές συναρτήσεις κ.λ.π. Τη σύνδεση αυτή την κάνει ένα ειδικό πρόγραμμα που ονομάζεται συνδέτης (linker). Το πρόγραμμα που παράγεται από το συνδέτη ονομάζεται εκτελέσιμο (executableprogram)

iv. Εκτελούμε το πρόγραμμα και βλέπουμε τις απαντήσεις στην οθόνη

Σχηματικά τα βήματα αυτά φαίνονται στο ακόλουθο διάγραμμα:



2 Αλφάβητο γλώσσας

Όπως κάθε ομιλούμενη γλώσσα χρησιμοποιεί τους χαρακτήρες του αλφαβήτου για να σχηματίσει λέξεις, έτσι και κάθε γλώσσα προγραμματισμού έχει το δικό της αλφάβητο. Με τους χαρακτήρες του αλφαβήτου της η γλώσσα σχηματίζει τις λέξεις ή τα σύμβολα που αποτελούν το λεξιλόγιο της γλώσσας. Το αλφάβητο της γλώσσας C περιλαμβάνει τους ακόλουθους χαρακτήρες:

- οριζόντιος στηλοθέτης (αναπαριστάνεται με το σύμβολο \t)
- κατακόρυφος στηλοθέτης (αναπαριστάνεται με το σύμβολο \v)
- αλλαγής σελίδας
- νέας γραμμής (αναπαριστάνεται με το σύμβολο \n)
- γράμματα αγγλικής αλφαβήτου κεφαλαία και πεζά

- αριθμοί
- σύμβολα πράξεων + - * / %
- σύμβολα στίξης ( ) & { } [ ] , . κ.λ.π.

3 Λεξιλόγιο γλώσσας

Το λεξιλόγιο της C περιλαμβάνει τα ακόλουθα:

- δεσμευμένες λέξεις (reserved words)
- τελεστές (operators)
- αναγνωριστές (identifiers)

3.1 Δεσμευμένες λέξεις

Οι δεσμευμένες λέξεις μιας γλώσσας έχουν αυστηρά καθορισμένη έννοια και τρόπο χρήσης και χρησιμοποιούνται σε συγκεκριμένες θέσεις ενός προγράμματος. Μερικές από τις πιο χαρακτηριστικές δεσμευμένες λέξεις είναι οι εξής:

- ***int***, ***float***, ***do***, ***if***, ***for***, ***sizeof*** κ.λ.π.

3.2 Αναγνωριστές

Στην κατηγορία των αναγνωριστών ανήκουν λέξεις που κατασκευάζει ο προγραμματιστής. Τέτοιες λέξεις δίνει ο προγραμματιστής σε μεταβλητές, σταθερές. συναρτήσεις και δικούς του τύπους δεδομένων. Η δημιουργία ονομάτων στη C διέπεται από ένα σύνολο κανόνων οι οποίοι είναι ανεξάρτητοι από τον τύπου του αντικειμένου στον οποίο αναφέρεται το όνομα. Αυτοί οι κανόνες αναφέρονται στο κεφάλαιο

- το όνομα πρέπει να περιλαμβάνει το πολύ 31 χαρακτήρες

4 Μεταβλητές-Σταθερές

Μεταβλητή (variable) είναι μια ποσότητα η οποία μπορεί να μεταβάλλει την τιμή κατά τη διάρκεια εκτέλεσης ενός προγράμματος, ενώ σταθερά (constant) είναι μια ποσότητα η οποία έχει σταθερή τιμή σε όλη τη διάρκεια εκτέλεσης του προγράμματος. Μεταβλητές χρησιμοποιούνται σχεδόν σε κάθε πρόγραμμα (πλην ελαχίστων εξαιρέσεων) και είναι θέσεις μνήμης στις οποίες:

- εισάγουμε δεδομένα από το πληκτρολόγιο
- καταχωρούμε αποτελέσματα που υπολογίζουμε από το πρόγραμμα

Τα ονόματα των μεταβλητών επιλέγονται ελεύθερα από τον προγραμματιστή πρέπει όμως υποχρεωτικά να υπακούουν στους ακόλουθους κανόνες:

i. Ο αρχικός χαρακτήρας πρέπει να είναι γράμμα ή underscore (_)
ii. Οι υπόλοιποι χαρακτήρες μπορεί να είναι γράμματα, αριθμοί ή underscore(_)
iii. Το μέγιστο μέγεθος μιας μεταβλητής μπορεί να είναι 32 χαρακτήρες
iv. Παίζει ρόλο ο τρόπος γραφής πεζών-κεφαλαίων. Έτσι τα ονόματα rate, RATE και Rate είναι 3 διαφορετικές μεταβλητές
v. Δεν μπορούν να χρησιμοποιούνται δεσμευμένες λέξεις της γλώσσας ως μεταβλητές

Όλες οι μεταβλητές που χρησιμοποιούνται σε ένα πρόγραμμα πρέπει υποχρεωτικά να δηλώνονται πριν τη χρήση τους γιατί αλλιώς θα έχουμε συντακτικό λάθος. Η δήλωση μιας μεταβλητής γίνεται με την ακόλουθη εντολή:

τύπος όνομα μεταβλητής;

Παραδείγματα δηλώσεων

int x	η μεταβλητή x δηλώνεται ακέραιου τύπου
float y	η μεταβλητή y δηλώνεται πραγματικού τύπου
char z	η μεταβλητή z δηλώνεται τύπου χαρακτήρα
double k	η μεταβλητή k δηλώνεται τύπου ***double***

Όλες οι μεταβλητές σε ένα πρόγραμμα σε C πρέπει να δηλώνονται πριν από τη χρήση τους σε κάποιο τύπο δεδομένων. Ο τύπος δεδομένων καθορίζει το σύνολο τιμών από το οποίο μπορεί να πάρει τιμές μια μεταβλητή. Οι τύποι δεδομένων διακρίνονται σε απλούς ή ενσωματωμένους και σε σύνθετους.

4.1 Τύποι Δεδομένων στη C

Οι τύποι δεδομένων δείχνουν το σύνολο τιμών από το οποίο μπορεί να πάρει τιμές μια μεταβλητή. Στη γλώσσα C υπάρχουν οι ακόλουθοι τύποι:

Τύπος	Μνήμη	Περιγραφή
int	2 bytes	Προσημασμένες ακέραιες τιμές από –32.768 έως +32.767
shortint ή ***short***	1 byte	Προσημασμένες ακέραιες τιμές από –128 έως +127
longint ή ***long***	4 bytes	Προσημασμένες ακέραιες τιμές από –2.147.483.648 έως +2.147.483.647
unsignedint ή ***unsigned***	2 bytes	Απρόσημες ακέραιες τιμές από 0 έως 65.535
char	1 byte	Το σύνολο των χαρακτήρων. χαρακτήρας αυτός μπαίνει μέσα σε αποστρόφους π.χ. 'A', '!' κ.λ.π.
signedchar	1 byte	Ένας χαρακτήρας με τιμές από –128 έως 127
unsignedchar	1 byte	Ένας χαρακτήρας με τιμές από 0 έως 255
float	4 bytes	Δεκαδικές τιμές από 3.4E-38 έως 3.4E+38 και –3.4E-38 έως -3.4E+38
double	8 bytes	Δεκαδικές τιμές από 1.7E-308 έως 1.7E+308 και –1.7E-308 έως -1.7E+308

5 Δομή προγράμματος σε C

Ένα πρόγραμμα σε C γράφεται με την ακόλουθη σειρά:

#***include***<***stdio.h***>	**à** ενσωματώνει στο πρόγραμμα μας το αρχείο επικεφαλίδας ***stdio.h***
voidmain()	**à** δήλωση προγράμματος (επικεφαλίδα προγράμματος)
{	**à** έναρξη προγράμματος
δήλωση μεταβλητών	
εντολή 1;	
εντολή 2;	
……	
εντολή n;	
}	**à** τέλος προγράμματος

5.1 Βασικοί Κανόνες της Γλώσσας

Στη γλώσσα C ισχύουν οι ακόλουθοι κανόνες:

· Κάθε πρόγραμμα στη γλώσσα C ξεκινά με τη συνάρτηση ***main***(). Η συνάρτηση αυτή είναι μοναδική και μπορεί να βρίσκεται οπουδήποτε μέσα σε ένα πρόγραμμα.

· Κάθε πρόγραμμα ξεκινά και τελειώνει με τα σύμβολα { και }

· Κάθε εντολή τελειώνει με ένα ελληνικό ερωτηματικό (;)

· Η συνάρτηση ***main***() πρέπει να επιστρέφει μια τιμή που να αντανακλά την κατάσταση του προγράμματος. Όταν επιστρέφεται η τιμή 0 στο λειτουργικό σύστημα τότε αυτό είναι ένδειξη ότι το πρόγραμμα τερματίζεται κανονικά χωρίς λάθη

6 Εντολές Εισόδου – Εξόδου

6.1 Εντολή Εισόδου

Με την εντολή εισόδου μπορούμε να εισάγουμε τιμές σε μεταβλητές. Έχει την ακόλουθη μορφή:

scanf("προσδιοριστής", &μεταβλητή);

Το σύμβολο & συμβολίζει τη διεύθυνση της μεταβλητής. Ο προσδιοριστής δηλώνει τι είδους περιεχόμενο μπορούμε να εισάγουμε στη μεταβλητή.

6.2 Προσδιοριστές για την *scanf*

Προσδιοριστής	Έξοδος
%c	ερμηνεύει την είσοδο ως χαρακτήρα (character)
%s	ερμηνεύει την είσοδο ως συμβολοσειρά (string). Η είσοδος αρχίζει με τον πρώτο χαρακτήρα που δεν είναι κενό διάστημα και τελειώνει στον πρώτο κενό χαρακτήρα
%d	ερμηνεύει την είσοδο ως προσημασμένο δεκαδικό ακέραιο (decimal)
%u	ερμηνεύει την είσοδο ως μη προσημασμένο δεκαδικό ακέραιο (***unsigned***)
%f ή %e	ερμηνεύει την είσοδο ως αριθμό κινητής υποδιαστολής (***float***)

Παραδείγματα

scanf("%d",&x)	εισάγουμε μια ακέραια προσημασμένη τιμή
scanf("%d",&y)	εισάγουμε μια δεκαδική προσημασμένη τιμή
scanf("%c",&z)	εισάγουμε ένα χαρακτήρα
scanf("%s",&k)	εισάγουμε μια συμβολοσειρά

6.3 Εντολή Εξόδου

Με την εντολή εξόδου μπορούμε να εμφανίσουμε στην οθόνη τις τιμές των μεταβλητών και σχόλια. Έχει την ακόλουθη μορφή:

printf ("σχόλιο");

ή

printf ("προσδιοριστής", μεταβλητή);

Ο προσδιοριστής καθορίζει το πώς θα εμφανιστούν τα δεδομένα σε αναγνώσιμη μορφή.

6.4 Προσδιοριστές για την *printf*

Προσδιοριστής	Έξοδος
%c	απλός χαρακτήρας (character)
%s	συμβολοσειρά (string)
%d	δεκαδικός ακέραιος με πρόσημο (decimal)
%u	δεκαδικός ακέραιος χωρίς πρόσημο (***unsigned***)
%f	αριθμός κινητής υποδιαστολής με δεκαδικό συμβολισμό (***float***)
%e	αριθμός κινητής υποδιαστολής με εκθετικό συμβολισμό (exponential)

Παραδείγματα

printf ("The result is")	εμφανίζουμε ένα σχόλιο
printf ("The sum is %d",sum)	εμφανίζουμε ένα σχόλιο και το περιεχόμενο της μεταβλητής sum σαν ακέραια τιμή
printf ("The difference is %4d",dif)	εμφανίζουμε ένα σχόλιο και το περιεχόμενο της μεταβλητής dif σαν ακέραια τιμή με πλάτος 4 θέσεις
printf("The average is %5.2f",aver)	εμφανίζουμε ένα σχόλιο και το περιεχόμενο της μεταβλητής aver σαν πραγματική τιμή με πλάτος 5 θέσεις για το ακέραιο μέρος και 2 θέσεις για το δεκαδικό μέρος

6.5 Λοιπές Συναρτήσεις Εισόδου/Εξόδου

puts("κείμενο") ή puts(μεταβλητή τύπου πίνακα χαρακτήρων δηλ. συμβολοσειρά)
Τυπώνει ένα κείμενο στην οθόνη ή μια μεταβλητή τύπου συμβολοσειράς
Ισοδύναμο με την εντολή printf("κείμενο\n");

Παράδειγμα
puts("dosearithmo"); ή ***printf***("dosearithmo\n");

gets(μεταβλητή τύπου πίνακα χαρακτήρων δηλ. συμβολοσειρά);
Διαβάζει ένα πίνακα χαρακτήρων
Ισοδύναμο με την εντολή scanf("%s",μεταβλητή τύπου πίνακα χαρακτήρων);

Παράδειγμα
char st[10];
gets(st); ή***scanf***("%s", st);

getchar();
Διαβάζει ένα χαρακτήρα
Ισοδύναμο με την εντολή ***scanf***("%c",&μεταβλητή τύπου χαρακτήρα");

Παράδειγμα
charx;
getchar(x); ή ***scanf***("%c", &x);

putchar();
Τυπώνει ένα χαρακτήρα
Ισοδύναμο με την εντολή ***printf***("%c",μεταβλητή τύπου χαρακτήρα");

Παράδειγμα
charx='a';
putchar(x); ή ***printf***("%c", x);

6.6 Εντολή #*include*

Γράφοντας την εντολή:

- #***include***<filename> ψάχνει για το αρχείο που καθορίζουμε μόνο στον ειδικό κατάλογο για ***include*** files
- #***include*** "filename" ψάχνει για το αρχείο που καθορίζουμε και στον τρέχοντα κατάλογο αλλά και στο ***include***directory

6.7 Εντολή #*define* και const

Γράφοντας την εντολή:

- #***define***DAYS_IN_WEEK 7 ορίζουμε τη σταθερά DAYS_IN_WEEK να παίρνει την τιμή 7. Αυτές οι σταθερές κληρονομούνται από τη γλώσσα C και ονομάζονται macro-based σταθερές. Οι σταθερές αυτές δηλώνονται πάντα έξω από το ***main***()
- const ***int*** DAYS_IN_WEEK=7; ορίζουμεμια formal constant. Ορίζοντας μια formal constant μέσα στο ***main***() έχει διαφορά από το να δηλωθεί έξω από το ***main***() διότι στην πρώτη περίπτωση η σταθερά αυτή είναι τοπική (local) για τη συνάρτηση ***main***() ενώ στη δεύτερη περίπτωση η σταθερά είναι γενική (global) δηλαδή διαθέσιμη σε οποιαδήποτε άλλη συνάρτηση του προγράμματος. Αυτή δήλωση γίνεται μόνο στη C++.

6.8 Τελεστές μοναδιαίας αύξησης και μείωσης

Οι τελεστές (μοναδιαίας) αύξησης έχουν τη μορφή:

- **μεταβλητή++** όπου πρώτα χρησιμοποιείται η τιμή της μεταβλητής στην έκφραση και μετά αυξάνεται. Ο τελεστής αυτός ονομάζεται επιθεματικός.
- **++μεταβλητή** όπου πρώτα αυξάνεται η τιμή της μεταβλητής και μετά χρησιμοποιείται στην έκφραση. Ο τελεστής αυτός ονομάζεται προθεματικός
- **μεταβλητή--** όπου πρώτα χρησιμοποιείται η τιμή της μεταβλητής στην έκφραση και μετά μειώνεται. Ο τελεστής αυτός ονομάζεται επιθεματικός
- **--μεταβλητή** όπου πρώτα μειώνεται η τιμή της μεταβλητής και μετά χρησιμοποιείται στην έκφραση. Ο τελεστής αυτός ονομάζεται προθεματικός

6.9 Τελεστές αύξησης και μείωσης

Οι τελεστές αύξησης και μείωσης έχουν τη μορφή:

- **μεταβλητή+= τιμή** η μεταβλητή αυξάνεται κατά μια συγκεκριμένη τιμή. Η εντολή αυτή μπορεί να γραφεί ισοδύναμα ως εξής: μεταβλητή= μεταβλητή + τιμή;
- **μεταβλητή-= τιμή** η μεταβλητή μειώνεται κατά μια συγκεκριμένη τιμή. Η εντολή αυτή μπορεί να γραφεί ισοδύναμα ως εξής: μεταβλητή= μεταβλητή - τιμή;
- **μεταβλητή*= τιμή** η μεταβλητή πολλαπλασιάζεται με μια συγκεκριμένη τιμή. Η εντολή αυτή μπορεί να γραφεί ισοδύναμα ως εξής: μεταβλητή= μεταβλητή * τιμή;
- **μεταβλητή/=τιμή** η μεταβλητή διαιρείται με μια συγκεκριμένη τιμή. Η εντολή αυτή μπορεί να γραφεί ισοδύναμα ως εξής: μεταβλητή= μεταβλητή / τιμή;

Παραδείγματα

y= x++;	Η εντολή αυτή είναι ισοδύναμη με τις εντολές y=x; x=x+1;
y= ++x;	Η εντολή αυτή είναι ισοδύναμη με τις εντολές x=x+1; y=x;
y= x--;	Η εντολή αυτή είναι ισοδύναμη με τις εντολές y=x; x=x-1;
y=--x;	Η εντολή αυτή είναι ισοδύναμη με τις εντολές x=x-1; y=x;
x+=2;	Η εντολή αυτή είναι ισοδύναμη με την x=x+2;
x-=3;	Η εντολή αυτή είναι ισοδύναμη με την x=x-3;
x*=4;	Η εντολή αυτή είναι ισοδύναμη με την x=x*4;
x/=5;	Η εντολή αυτή είναι ισοδύναμη με την x=x/5;

6.10 Τελεστής *sizeof*

Ο τελεστής ***sizeof*** έχει την ακόλουθη σύνταξη: ***sizeof***(μεταβλητή) και δίνει το μέγεθος της μεταβλητής αυτής σε bytes.

7 Ανάλυση βασικών τύπων της C

7.1 Ο τύπος του χαρακτήρα

Ο τύπος χαρακτήρα χρησιμοποιείται για να αναπαραστήσει απλούς χαρακτήρες του αλφαβήτου της γλώσσας και δηλώνεται με τη λέξη ***char***. Μια σταθερά τύπου χαρακτήρα εμφανίζεται σε ένα πρόγραμμα μέσα σε απλά εισαγωγικά π.χ. 'C', '2' κ.λ.π. Για τη δήλωση μιας μεταβλητής τύπου χαρακτήρα ακολουθείται ο γενικός κανόνας δήλωσης μεταβλητών. Έτσι για να δηλώσουμε μια μεταβλητή χαρακτήρα με όνομα answer και αρχική τιμή το γράμμα Α γράφουμε την παρακάτω πρόταση:

char answer='A';

Για να εκτυπώσουμε μια μεταβλητή χαρακτήρα χρησιμοποιούμε όπως αναφέραμε τον προσδιοριστή %c π.χ. για να εκτυπώσουμε τη μεταβλητή answer γράφουμε την εντολή:

printf("Ο χαρακτήρας είναι %c\n", answer);

Για να εισάγουμε τιμή σε μεταβλητή τύπου χαρακτήρα χρησιμοποιούμε την εντολή ***scanf***. Π.χ. για να δώσουμε τιμή στη μεταβλητής answer γράφουμε την εντολή:

scanf("%c", &answer);

7.2 Μη εκτυπώσιμοι χαρακτήρες

Οι μη εκτυπώσιμοι χαρακτήρες της C είναι οι ακόλουθοι:

Χαρακτήρας	**Αναπαράσταση**
νέα γραμμή	\n
οριζόντιος στηλοθέτης	\t
κατακόρυφος στηλοθέτης	\v
backspace	\b
slush	\\
ερωτηματικό	\?
μονό εισαγωγικό	\'
διπλό εισαγωγικό	\"

Παράδειγμα

Το επόμενο πρόγραμμα διαβάζει ένα χαρακτήρα και τυπώνει το χαρακτήρα αυτό, τον επόμενο του καθώς και τους ASCII κωδικούς τους.

```
#include<stdio.h>

voidmain()
{
char ch, next_ch;

      printf("Δώσεχαρακτήρα \n");
      scanf("%c", &ch);
      printf("Ο ASCII κωδικός του χαρακτήρα είναι %c είναι %d \n",ch, ch);
      next_ch=ch+1;
      printf("Ο ASCII κωδικός του χαρακτήρα είναι %c είναι %d \n",ch, ch);
}
```

7.3 Ο τύπος του ακεραίου

Ο τύπος του ακέραιου που δηλώνεται με τη λέξη ***int*** χρησιμοποιείται για να αναπαραστήσει ακέραιους αριθμούς. Οι ακέραιοι αριθμοί μπορεί να είναι θετικοί ή αρνητικοί και το εύρος των τιμών που μπορούν να πάρουν εξαρτάται από την αρχιτεκτονική του Η/Υ επειδή για την αποθήκευση τους χρησιμοποιούνται τόσα bits όσο είναι και το μήκος της λέξης του συγκεκριμένου επεξεργαστή. Έτσι για ένα Η/Υ με μήκος λέξης 16 bit η περιοχή τιμών του ***int*** είναι από –32767 μέχρι 32768 για προσημασμένους ακέραιους και 0 μέχρι 65535 για απρόσημους. Ο προσδιοριστής ***long*** χρησιμοποιείται πριν από τη λέξη ***int*** για να προσδιορίσει ακέραιο με μεγαλύτερο εύρος τιμών. Αντίστοιχα ο προσδιοριστής ***unsigned*** χρησιμοποιείται στη δήλωση πριν από τη λέξη ***int*** για να χαρακτηρίσει τη μεταβλητή ως απρόσημη.

Έτσι για να δηλώσουμε μια μεταβλητή ακέραια με όνομα number και αρχική τιμή τον αριθμό 5 γράφουμε την παρακάτω πρόταση:

int number=5;

Για να εκτυπώσουμε μια ακέραια μεταβλητή χρησιμοποιούμε όπως αναφέραμε τον προσδιοριστή %d π.χ. για να εκτυπώσουμε τη μεταβλητή number γράφουμε την εντολή:

printf("Ο αριθμός είναι %d \n", number);

Για να εισάγουμε τιμή σε ακέραια μεταβλητή χρησιμοποιούμε την εντολή ***scanf***. Π.χ. για να δώσουμε τιμή στη μεταβλητή number γράφουμε την εντολή:

scanf("%d", &number);

Σημείωση

Όταν γράφουμε ένα αριθμό στον πηγαίο κώδικα χωρίς δεκαδικό ή εκθετικό μέρος ο μεταγλωττιστής τον χειρίζεται σαν ακέραια σταθερά. Έτσι η σταθερά 245 αποθηκεύεται σαν ***int***, ενώ η σταθερά 100.000 αποθηκεύεται σαν ***long***. Για ελεγχόμενη αποθήκευση χρησιμοποιούμε τους προσδιοριστές l ή L μετά τον αριθμό. Η σταθερά 8965L αναγκάζει το μεταγλωττιστή να δεσμεύσει χώρο για ***longint***.

7.4 Ο τύπος του πραγματικού

Πραγματικοί είναι οι αριθμοί που διαθέτουν κλασματικό μέρος. Οι πραγματικοί αριθμοί εκφράζονται συνήθως σε fixed point, floating point ή εκθετική μορφή όπως φαίνεται στον ακόλουθο πίνακα:

fixed-point number	**floating-point number**	**εκθετική μορφή**
123.456	$1.23456X10^{2}$	1.23456e+02
0.00002	$2.0X10^{-5}$	$2.0e^{-5}$
50000.0	$5.0X10^{4}$	5.0e+04

Η C διαθέτει για περισσότερη ευελιξία 2 τύπους για την αναπαράσταση των πραγματικών αριθμών. Τον τύπο ***float*** για αριθμούς κινητής υποδιαστολής απλής ακρίβειας και τον τύπο ***double*** για τους αριθμούς κινητής υποδιαστολής διπλής ακρίβειας.

8 Εντολές Επιλογής

8.1 Απλή επιλογή

Στην απλή επιλογή ελέγχεται μια συνθήκη και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 1, ενώ αν είναι ψευδής τότε εκτελείται μια ομάδα εντολών 2. Έχει την ακόλουθη σύνταξη:

```
if (συνθήκη)
{
   ομάδα εντολών 1;
}
else
{
     ομάδα εντολών 2;
}
```

Παραδείγματα

```
int b;
if (b>5)
{
   printf(“Success \n”);
}
else
{
   printf(“Failure \n”);
}

int s,y;
if (s>0)
{
   y=s;
}
else
{
   y=-s;
}
printf("The absolute value of %d is %d \n",s,y);
```

8.2 Περιορισμένη επιλογή

Στην περιορισμένη επιλογή ελέγχεται μια συνθήκη και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 1 ενώ αν είναι ψευδής τότε δεν εκτελείται τίποτα και το πρόγραμμα συνεχίζει απλώς στην επόμενη εντολή μετά το ***if***. Έχει την ακόλουθη σύνταξη:

```
if (συνθήκη)
{
   ομάδα εντολών 1;
}
```

Παραδείγματα

```
int b;
if (b>5)
{
   printf(“Success \n”);
```

```
}
if (b<=5)
{
   printf("Failure \n");
}


int b;
if (b%2==0)
{
   printf("Even number \n");
}
if (b%2==1)
{
   printf("Odd number \n");
}
```

8.3 Εμφωλευμένη επιλογή

Στην εμφωλευμένη επιλογή ελέγχεται μια συνθήκη 1 και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 1. Αν είναι ψευδής τότε ελέγχεται μια συνθήκη 2 και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 2. Αν είναι ψευδής τότε ελέγχεται μια συνθήκη 3 κ.ο.κ. Συνολικά ελέγχονται n συνθήκες και εκτελείται μόνο μια ομάδα εντολών. Αν δεν είναι καμία συνθήκη αληθής τότε εκτελείται μια ομάδα εντολών n+1. Έχει την ακόλουθη σύνταξη:

```
if (συνθήκη 1)
{
  ομάδα εντολών 1;
}
else
      if (συνθήκη 2)
      {
         ομάδα εντολών 2;
      }
      else
      ……..
            if (συνθήκη n)
            {
               ομάδα εντολών n;
            }
            else
                  {
                    ομάδα εντολών n+1;
                  }
```

Παράδειγμα 1

```
int b;
if (b>0)
{
   printf("Positive \n");
}
elseif (b<0)
```

```
{
        printf("Negative \n");
}
    else
    {
        printf("Zero \n");
    }
```

Παράδειγμα 2

```
int a,b;
if (a>b)
{
   printf("Greater umber is \n",a);
}
elseif (a<b)
{
        printf("Greater number is\n",b);
}
      else
      {
            printf("Equal numbers \n");
      }
```

Σημειώσεις

1.Σε κάθε εντολή επιλογής η συνθήκη είναι μια λογική παράσταση η οποία παίρνει είτε την τιμή αλήθεια (true) είτε την τιμή ψέμα (false). Το 0 αντιστοιχεί στη λογική τιμή αλήθεια και ένας αριθμός ¹ 0 αντιστοιχεί στην λογική τιμή ψέμα.

2.Κάθε ομάδα εντολών μπορεί να είναι είτε μια μόνο εντολή ή ένα μπλοκ από εντολές μέσα σε άγκιστρα. Αν είναι μόνο μια εντολή τότε μπορούμε να παραλείψουμε τα άγκιστρα.

8.4 Τριαδικός τελεστής

Ο τριαδικός τελεστής είναι μια πράξη παράγει δηλαδή μια τιμή σαν αποτέλεσμα. Ο τριαδικός τελεστής επιλέγει μια από δύο δυνατές τιμές ανάλογα με την τιμή μιας λογικής συνθήκης:

συνθήκη?τιμή1:τιμή2

Αν η συνθήκη είναι αληθής επιλέγεται η τιμή1 αλλιώς επιλέγεται η τιμή2. Ο τριαδικός τελεστής αντικαθιστά ουσιαστικά εντολές ***if***.

Παράδειγμα

z=(x>y)?5:10

δηλαδή αν το x>y τότε στο z καταχωρείται η τιμή 10 αλλιώς καταχωρείται η τιμή 5.

8.5 Εντολή πολλαπλής επιλογής *switch*

Η εντολή πολλαπλής επιλογής χρησιμοποιείται όταν ελέγχουμε την τιμή μιας μεταβλητής (όχι συνθήκης) και ανάλογα με την τιμή αυτή εκτελούμε είτε μια ομάδα εντολών 1 είτε μια ομάδα εντολών 2 είτε μια εντολών n. Αν η μεταβλητή δεν πάρει καμία από τις τιμές 1, 2, … n τότε εκτελείται η ομάδα εντολών που βρίσκεται στο ***default***. Η εντολή ***switch*** γενικά έχει την ακόλουθη μορφή:

```
switch (μεταβλητή)
{
   case τιμή 1:      ομάδα εντολών 1;
                     break;
   case τιμή 2:      ομάδα εντολών 2;
                     break;
                     ……..
   case τιμή n:      ομάδα εντολών n;
                     break;
   default:          ομάδα εντολών n+1;
}
```

Σημείωση

Η εντολή ***break*** είναι προαιρετική. Όμως ουσιαστικά είναι επιβεβλημένη η χρησιμοποίηση της γιατί αν απουσιάζει από το τέλος εντολών μιας ομάδας τότε εκτελούνται οι εντολές και της επόμενης ομάδας γεγονός βέβαια που δεν έχει νόημα. Συνεπώς η ***break*** στην εντολή ***switch*** απαγορεύει την εκτέλεση των εντολών των υπολοίπων ομάδων πέρα της ομάδας εντολών που θα εκτελεστεί.

Παράδειγμα 1

```
int x;
switch (x)
{
   case 4:    printf("Failure \n");
              break;
   case 5:    printf("Base \n");
              break;
   case 6:    printf("Good \n");
              break;
   case 8:    printf("Very Good \n");
              break;
   case 10:   printf("Excellent \n");
              break;
   default:   printf("Invalid Grade \n");
}
```

Παράδειγμα 2

```
char operator;
int sum,dif,prod,div,rem;
switch (operator)
{
   case '+':  sum=a+b;
              printf("The sum is %d \n", sum);
              break;
   case '-':  dif=a-b;
              printf("The difference is %d \n", dif);
              break;
   case '*':  prod=a*b;
              printf("The product is %d \n", prod);
              break;
```

```
        case '/':   div=a/b;
                    printf("The quotient is %d \n", div);
                    break;
        case '%':   rem=a%b;
                    printf("The remainder is %d \n", rem);
                    break;
        default:    printf("Invalid operator \n");
    }
```

9 Εντολές Επανάληψης

Οι εντολές επανάληψης (ανακύκλωσης) προκαλούν την εκτέλεση μιας εντολής ή μιας ομάδας εντολών μέσα σε άγκιστρα είτε ένα προκαθορισμένο αριθμό επαναλήψεων είτε για όσο χρόνο μια συνθήκη είναι αληθής. Στη C χρησιμοποιούμε 3 εντολές επανάληψης.

9.1 Εντολή επανάληψης *for*

Η εντολή ***for*** επαναλαμβάνει μια ομάδα εντολών συνήθως ένα συγκεκριμένο αριθμό φορών αν και μπορεί να χρησιμοποιηθεί ακόμα και για την επανάληψη μιας ομάδας εντολών για όσο χρόνο μια συνθήκη είναι αληθής. Έχει την ακόλουθη σύνταξη:

```
for (αρχικές τιμές; συνθήκη; μεταβολές)
{
    ομάδα εντολών;
}
```

Οι αρχικές τιμές μπορεί να είναι μια ή περισσότερες εντολές που διαχωρίζονται με κόμμα και έχουν στόχο να εκτελεστούν κάποιες εντολές πριν αρχίσει η επανάληψη. Η συνθήκη μπορεί να είναι απλή είτε να συγκροτείται από επιμέρους συνθήκες χωριζόμενες με κόμμα (σύνθετη) και για όσο χρόνο είναι αληθής εκτελείται η ομάδα εντολών ανάμεσα στα άγκιστρα. Μετά από κάθε επανάληψη εκτελούνται οι εντολές που βρίσκονται στο τμήμα μεταβολές οι οποίες έχουν σαν σκοπό να μεταβάλλουν την τιμή της συνθήκης. Στη συνέχεια ελέγχεται πάλι η νέα τιμή της συνθήκης και εφόσον εξακολουθεί να είναι αληθής επαναλαμβάνεται η εκτέλεση της ομάδας εντολών. Όταν η συνθήκη γίνει ψευδής τότε η εντολή ***for*** τερματίζεται και το πρόγραμμα συνεχίζει στην επόμενη εντολή μετά την ***for***.

Παραδείγματα εντολής ***for***

Στο ακόλουθο παράδειγμα ο αριθμός των επαναλήψεων είναι προκαθορισμένος καθώς η ομάδα εντολών ανάμεσα στα άγκιστρα θα εκτελεστεί 5 φορές:

```
sum=0;
for (i=1;i<=5;i++)
{
        printf("Give a grade \n");
        scanf("%d",&grade);
        sum+=grade;
}
av=grade/5;
printf("the average grade is %.2f \n",av);
```

Στο ακόλουθο παράδειγμα ο αριθμός των επαναλήψεων δεν είναι προκαθορισμένος. Η επανάληψη εκτελείται μέχρι να δοθεί αρνητικός αριθμός

```
counter=0;
```

```
for (x=0; x>=0 ;counter++)
{
        printf("Give a number \n");
        scanf("%d",&x);
}
printf("%d positive numbers were given \n", counter);
```

9.2 Εντολή Επανάληψης *while*

Η εντολή ***while*** χρησιμοποιείται όπως και η ***for*** για να επαναλάβουμε την εκτέλεση μιας εντολής ή μιας ομάδας εντολών μέσα σε άγκιστρα είτε ένα συγκεκριμένο αριθμό φορών είτε για όσο χρόνο μια συνθήκη είναι αληθής. Πρώτα ελέγχεται η συνθήκη και εφόσον είναι αληθής τότε εκτελείται η ομάδα εντολών. Για όσο χρόνο η συνθήκη (απλή ή σύνθετη) είναι αληθής (true) τότε επαναλαμβάνεται η εκτέλεση της ομάδας εντολών. Όταν η συνθήκη γίνει ψευδής η εντολή ***while*** τερματίζεται και το πρόγραμμα συνεχίζει στην επόμενη εντολή μετά την ***while***. Έχει την ακόλουθη σύνταξη:

```
while (συνθήκη)
 {
    ομάδα εντολών;
 }
```

<u>Παραδείγματα εντολής *while*</u>

Στο ακόλουθο παράδειγμα ο αριθμός των επαναλήψεων είναι προκαθορισμένος καθώς η ομάδα εντολών ανάμεσα στα άγκιστρα θα εκτελεστεί n φορές:

```
s=f=0;
i=1;
printf("Give count of numbers");
scanf("%d",&n);
while (i++<=n)
{
        printf("Give a grade \n");
        if (grade>10)
        {
                s++;
        }
        else
        {
                f++;
        }
}
printf("Successful grades are %d \n", s);
printf("Unsuccessful grades are %d \n", f);
```

Στο ακόλουθο παράδειγμα ο αριθμός των επαναλήψεων δεν είναι προκαθορισμένος. Η επανάληψη εκτελείται μέχρι να δοθεί το –1 ή το 1

```
counter=0;
printf("Give initial number \n");
scanf("%d",&x);
while (x=!1 && x!=-1)
```

```
{
        counter++;
        printf("Give next number \n");
        scanf("%d",&x);
}
printf("%d numbers were given until –1 or 1\n", counter);
```

9.3 Εντολή επανάληψης *do..while*

Η εντολή ***dowhile*** είναι αντίστοιχη με την εντολή ***while*** με τη διαφορά ότι πρώτα εκτελείται η ομάδα εντολών και μετά ελέγχεται η συνθήκη, δηλαδή η ομάδα εντολών θα εκτελεστεί τουλάχιστον μια φορά ακόμα και αν η συνθήκη είναι ψευδής. Έχει την ακόλουθη σύνταξη:

```
do
{
   ομάδα εντολών;
}while (συνθήκη);
```

Παραδείγματα εντολής *do..while*

Στο ακόλουθο παράδειγμα ο αριθμός των επαναλήψεων είναι προκαθορισμένος καθώς η ομάδα εντολών ανάμεσα στα άγκιστρα θα εκτελεστεί n φορές:

```
s=f=0;
i=1;
printf(“Give count of numbers”);
scanf(“%d”,&n);
do
{
        printf("Give a grade \n");
        if (grade>10)
        {
                s++;
        }
        else
        {
                f++;
        }
} while (i++<=n)
printf("Successful grades are %d \n", s);
printf("Unsuccessful grades are %d \n", f);
```

Στο ακόλουθο παράδειγμα ο αριθμός των επαναλήψεων δεν είναι προκαθορισμένος. Η επανάληψη εκτελείται μέχρι να δοθεί το –1 ή το 1

```
counter=0;
printf("Give initial number \n");
scanf("%d",&x);
do
{
        counter++;
        printf("Give next number \n");
```

```
        scanf("%d",&x);
} while (x=!1 && x!=-1);
printf("%d numbers were given until –1 or 1\n", counter);
```

10 Εντολή *break*

Στις εντολές επανάληψης ***for***, ***while*** και ***dowhile*** μπορούμε να παρεμβάλουμε μέσα στην ομάδα εντολών την εντολή ***break*** η οποία προκαλεί τη διακοπή της εκτέλεσης των υπολοίπων εντολών της ομάδας και κάνει άμεση έξοδο από την επανάληψη. Έχει την ακόλουθη σύνταξη:

```
Εντολή επανάληψης
{
   break;
}
```

Παράδειγμα εντολής *break*

Στο ακόλουθο παράδειγμα εφόσον δώσουμε βαθμό μη αποδεκτό (δηλαδή έξω από το διάστημα 0-20) τότε η εντολή ***break*** προκαλεί τον άμεσο τερματισμό της επανάληψης και την εκτύπωση των αποτελεσμάτων που έχουν υπολογιστεί μέχρι εκείνη τη στιγμή.

```
s=f=0;
i=1;
printf(“Give count of numbers”);
scanf(“%d”,&n);
do
{
        printf("Give a grade \n");
        scanf("%d",&grade);
        if (grade>=0 && grade<10)
        {
                s++;
        }
        elseif (grade>=10 && grade<=20)
        {
                f++;
        }
            else
            {
                 printf(“You gave an invalid grade\n”);
                 break;
            }
} while (i++<=n);
printf("Successful grades are %d \n", s);
printf("Unsuccessful grades are %d \n", f);
```

10.1 Εντολή *continue*

Η εντολή ***continue*** τοποθετείται επίσης μέσα σε μια επανάληψη και προκαλεί τη διακοπή της εκτέλεσης των υπολοίπων εντολών της ομάδας αλλά δεν κάνει έξοδο από το βρόχο, απλώς επιβάλλει επανέλεγχο της συνθήκης. Έχει την ακόλουθη σύνταξη:

```
Εντολή επανάληψης
{
   continue;
}
```

Παράδειγμα εντολής *continue*

Στο ακόλουθο παράδειγμα εφόσον δώσουμε βαθμό μη αποδεκτό (δηλαδή έξω από το διάστημα 0-20) τότε η εντολή ***continue*** αγνοεί τις υπόλοιπες εντολές του βρόχου και ξαναελέγχει τη συνθήκη:

```
s=f=0;
i=1;
printf("Give count of numbers");
scanf("%d",&n);
do
{
        printf("Give a grade \n");
        scanf("%d",&grade);
        if (grade<0 || grade>20)
            {
                printf("You gave an invalid grade\n");
                continue;
            }
            elseif (grade>=0 && grade<10)
                {
                    s++;
                }
                elseif (grade>=10 && grade<=20)
                    {
                        f++;
                    }
} while (i++<=n);
printf("Successful grades are %d \n", s);
printf("Unsuccessful grades are %d \n", f);
```

10.2 Τελεστές

10.2.1 Σχεσιακοί Τελεστές

Οι σχεσιακοί τελεστές χρησιμοποιούνται για να δημιουργήσουν συνθήκες. Στη γλώσσα C είναι οι ακόλουθοι:

Σχεσιακοί Τελεστές
< (μικρότερο)
<=(μικρότερο ή ίσο)
> (μεγαλύτερο)
>=(μεγαλύτερο ή ίσο)
== (ισότητα)
!= (διάφορο)

10.2.2 Παραδείγματα σχεσιακών τελεστών

x==3
y>=10
z!=0
a>0
b<20

Σημείωση

Πρέπει να τονίσουμε τη διαφορά ανάμεσα στην εντολή καταχώρισης η οποία συμβολίζεται με το = και τον τελεστή της ισότητας ο οποίος συμβολίζεται με ==.

10.2.3 Λογικοί Τελεστές

Οι λογικοί τελεστές χρησιμοποιούνται προκειμένου να συνδυάσουν πολλές απλές συνθήκες μέσα σε μια σύνθετη συνθήκη. Η τιμή μιας συνθήκης απλής ή σύνθετης είναι είτε true (αληθής) είτε false (ψευδής). Στη γλώσσα C οι λογικοί τελεστές είναι οι ακόλουθοι είναι οι ακόλουθοι:

Λογικοί Τελεστές
&& (AND)
|| (OR)
! (NOT)

α)λογικός τελεστής && (and)

Ο λογικός τελεστής && δίνει την τιμή true όταν όλες οι συνθήκες που ελέγχει είναι ταυτόχρονα αληθείς. Αν έστω και μια από τις συνθήκες που ελέγχει είναι ψευδής τότε δίνει την τιμή false. Η σύνταξη του είναι η ακόλουθη:

(συνθήκη1 && συνθήκη2 && συνθήκη3…..)

Ο πίνακας αλήθειας του λογικού τελεστή && για 2 συνθήκες φαίνεται στον ακόλουθο πίνακα:

Πίνακας Αλήθειας &&		
Σ1	**Σ2**	**ΑΠΟΤΕΛΕΣΜΑ**
Α	Α	Α

Α	Ψ	Ψ
Ψ	Α	Ψ
Ψ	Ψ	Ψ

β)λογικός τελεστής || (or)

Ο λογικός τελεστής || δίνει την τιμή true όταν έστω και μια από τις συνθήκες συνθήκη που ελέγχει είναι αληθής αλλιώς δίνει την τιμή false. Η σύνταξη του είναι η ακόλουθη:

(συνθήκη1 || συνθήκη2 || συνθήκη3…..)

Ο πίνακας αλήθειας του λογικού τελεστή || για 2 συνθήκες φαίνεται στον ακόλουθο πίνακα:

Πίνακας Αλήθειας ||		
Σ1	**Σ2**	**ΑΠΟΤΕΛΕΣΜΑ**
Α	Α	Α
Α	Ψ	Α
Ψ	Α	Α
Ψ	Ψ	Ψ

γ)λογικός τελεστής ! (not)

Ο λογικός τελεστής ! εφαρμόζεται σε μια μόνο συνθήκη και αντιστρέφει την τιμή της δηλαδή αν η συνθήκη είναι αληθής τότε δίνει την τιμή ψευδής (false) ενώ αν είναι ψευδής τότε δίνει την τιμή αληθής (true). Η σύνταξη του είναι η ακόλουθη:

!(συνθήκη)

Ο πίνακας αλήθειας του λογικού τελεστή ! φαίνεται στον ακόλουθο πίνακα:

Πίνακας Αλήθειας !	
Σ	**ΑΠΟΤΕΛΕΣΜΑ**
Α	Ψ
Ψ	Α

Σημείωση

Οι λογικοί τελεστές && και || μπορούν γενικά να εφαρμοστούν σε 2 ή περισσότερες συνθήκες. Η προτεραιότητα των λογικών τελεστών είναι κατά σειρά ! (not) μετά && (and) και τέλος || (or).

Παραδείγματα απλών και σύνθετων συνθηκών

Συνθήκη	**Χαρακτηρισμός**
if (x!=0)	Απλή συνθήκη. Έχει τιμή true αν το x είναι διάφορο του 0.
if (x>=0 && x<5)	Σύνθετη συνθήκη. Έχει τιμή true μόνο αν ικανοποιούνται και οι 2 συνθήκες.
if (x==9 || x==10)	Σύνθετη συνθήκη. Έχει τιμή true αν ικανοποιείται είτε η μια είτε η άλλη συνθήκη.
if !(x<0)	Απλή συνθήκη. Έχει τιμή true αν το x είναι μεγαλύτερο ή ίσο του 0.
while (x>=0 && x<=10) || (x>=20 && x<=30)	Εντολή επανάληψης που ελέγχει μια σύνθετη συνθήκη. Η επανάληψη εκτελείται αν η σύνθετη συνθήκη έχει την τιμή true ενώ σταματά η εκτέλεση της αν η σύνθετη συνθήκη έχει την τιμή false. Η σύνθετη συνθήκη έχει την τιμή true αν έστω και μια από τις 2 συνθήκες στην παρένθεση είναι true.

11 Πίνακες

Οι πίνακες είναι διαδοχικές θέσεις μνήμης στις οποίες τοποθετούνται δεδομένα του ιδίου τύπου. Στα δεδομένα ενός πίνακα αναφερόμαστε με ένα δείκτη θέσης (μονοδιάστατος πίνακας) ή με ένα ζεύγος δεικτών θέσης (δισδιάστατος πίνακας).

11.1 Μονοδιάστατοι πίνακες

Τα στοιχεία του πίνακα είναι όπως αναφέραμε του ιδίου τύπου και τοποθετούνται σε διαδοχικές θέσεις μνήμης. Λέγεται μονοδιάστατος ή γραμμικός πίνακας γιατί όλα τα στοιχεία του είναι τοποθετημένα σε μια γραμμή όπως φαίνεται και στο ακόλουθο σχήμα:

0	**1**	**2**	**3**	**4**	**5**
12	-7	45	7	4	8

Με τα έντονα γράμματα φαίνονται οι θέσεις του πίνακα. Παρατηρούμε ότι οι θέσεις ενός μονοδιάστατου πίνακα αριθμούνται αρχίζοντας από το 0.

11.1.1 Δήλωση μονοδιάστατου πίνακα

Η δήλωση ενός μονοδιάστατου πίνακα γίνεται με την ακόλουθη εντολή:

τύπος όνομα_πίνακα [πλήθος στοιχείων];

Για παράδειγμα γράφοντας σε ένα πρόγραμμα τη δήλωση ***int*** grades[5]; ορίζουμε ένα πίνακα με όνομα grades που συγκροτείται από 5 ακεραίους σε διαδοχικές θέσεις μνήμης.

grades[0]	grades[1]	grades[2]	grades[3]	grades[4]

Γενικά το στοιχείο ενός μονοδιάστατου πίνακα στη θέση i συμβολίζεται σαν Πίνακας[i]. Κάθε μία από τις διαδοχικές θέσεις μνήμης που συγκροτούν ένα πίνακα αποτελούν ένα στοιχείο του πίνακα.

grades[0]	Ο ακέραιος που υπάρχει στην πρώτη θέση του πίνακα grades	(1ο στοιχείο)
grades [1]	Ο ακέραιος που υπάρχει στην δεύτερη θέση του πίνακα grades	(2ο στοιχείο)
grades [2]	Ο ακέραιος που υπάρχει στην τρίτη θέση του πίνακα grades	(3ο στοιχείο)
grades [3]	Ο ακέραιος που υπάρχει στην τέταρτη θέση του πίνακα grades	(4οστοιχείο)
grades [4]	Ο ακέραιος που υπάρχει στην πέμπτη θέση του πίνακα grades	(5ο στοιχείο)

11.1.2 Αρχικές τιμές μονοδιάστατου πίνακα

Μαζί με την δήλωση ενός πίνακα μπορούμε να δώσουμε και αρχικές τιμές στα στοιχεία του πίνακα με εντολή της μορφής:

intnum[5]= {10, 25, 15, 35,5};

Τότε ο πίνακας num θα έχει τα ακόλουθα στοιχεία:

10	25	15	35	5

Δηλαδή θα ισχύουν: num[0]=10, num[1]=25, num[2]=15, num[3]=35, num[4]=5. Δηλώνοντας κάποιο πίνακα χωρίς αρχικές τιμές τα στοιχεία του έχουν τιμές απροσδιόριστες.

11.1.3 Προσαρμοζόμενη διάσταση μονοδιάστατου πίνακα

Όταν δηλώνουμε ένα πίνακα με αρχικές τιμές μπορούμε να μην προσδιορίζουμε το πλήθος των στοιχείων του αλλά να αφήνουμε την C να προσαρμόζει ανάλογα με το πλήθος των αρχικών τιμών και το πλήθος των στοιχείων του πίνακα. Τότε λέμε ότι ορίζουμε τον πίνακα με προσαρμοζόμενη διάσταση. Για παράδειγμα επιτρέπεται να δηλώσουμε ένα πίνακα με 5 στοιχεία γράφοντας:

```
int num[] = {10,25,15,35,5};
```

11.1.4 Εκχώρηση τιμών στα στοιχεία μονοδιάστατου πίνακα

Από τη στιγμή που έχουμε δηλώσει έναν πίνακα (με αρχικές τιμές ή χωρίς αρχικές τιμές) δεν υπάρχει τρόπος να εκχωρήσουμε, με μια εντολή, τιμές σε όλα τα στοιχεία του. Δηλαδή είναι λάθος να γράψουμε: num ={4, 8 12, 5, 20};
Οι τιμές καταχωρούνται μόνο σε μεμονωμένα στοιχεία του πίνακα κάθε φορά. Για παράδειγμα δίνοντας τις εντολές: num[0]=4; num[1]=8; num[2]=12; num[3]=5; num[4]=20; ο πίνακας num θα έχει τα ακόλουθα στοιχεία:

4	8	12	5	20

11.1.5 Διάβασμα μονοδιάστατου πίνακα

Για να διαβάσουμε (γεμίσουμε) ένα μονοδιάστατο πίνακα με τιμές γράφουμε τις ακόλουθες εντολές: (έστω ότι έχει προηγηθεί η δήλωση **float**grades[5];)

```
for (i=0;i<5;i++)
{
   printf("Give value to %d element\n",i+1);
   scanf("%d",&grades[i]);
}
```

11.1.6 Εκτύπωση μονοδιάστατου πίνακα

Για να εκτυπώσουμε (εμφανίσουμε) τα στοιχεία ενός μονοδιάστατου πίνακα στην οθόνη γράφουμε τις ακόλουθες εντολές: (έστω ότι έχει προηγηθεί η δήλωση **float**grades[5] και βέβαια ο πίνακας grades έχει πάρει τιμές)

```
for (i=0;i<5;i++)
{
   printf("%d",grades[i]);
}
```

11.1.7 Μέγεθος μονοδιάστατου πίνακα

Αν έχουμε ορίσει ένα πίνακα τότε το πλήθος όλων των bytes της μνήμης που αφιερώνονται για τα στοιχεία του πίνακα προσδιορίζεται μέσω του τελεστή ***sizeof***. Για παράδειγμα μετά την δήλωση ***int***num[]={1,2,3,4,5}; η εντολή ***printf***("Sizeofarraynuminbytesis %d\n",***sizeof***(num)); θα μας δώσει εκτυπώσει 10 (5 ακέραιοι επί 2 bytes ο καθένας).

Παράδειγμα: Διάβασμα μονοδιάστατου πίνακα εξασφαλίζοντας ότι λαμβάνει μοναδικές τιμές

```
#include<stdio.h>

#define N 5

voidmain()
{
        int x,b[5],i,j,found;

        for (i=0;i<N;i++)
        {
                if (i==0)
                {
                        printf("Dose 1o bathmo\n");
                        scanf("%d",&b[i]);
                }
                else
                {
                        do
                        {
                                found=0;

                                printf("Dose %do bathmo\n",i+1);
                                scanf("%d",&x);

                                for (j=0;j<i;j++)
                                        if (x==b[j])
                                        {
                                                printf("O bathmos %d yparxei hdh. \n",x);
                                                found=1;
                                        }
                                        else
                                                b[i]=x;
                        }
                        while (found==1);
                }
        }

        printf("\nOi bathmoi einai \n");
        for (i=0;i<N;i++)
                printf("%d\t",b[i]);
}
```

11.1.8 Δισδιάστατοι πίνακες

Τα στοιχεία του πίνακα είναι και πάλι του ιδίου τύπου και βρίσκονται επίσης σε διαδοχικές θέσεις μνήμης. Λέγεται δισδιάστατος ή ορθογώνιος πίνακας γιατί όλα τα στοιχεία του είναι τοποθετημένα σε γραμμές και στήλες όπως φαίνεται και στο ακόλουθο σχήμα:

	0	**1**	**2**	**3**	**4**
0	12	-7	45	7	4
1	14	8	10	6	60
2	-9	5	20	3	2

Με τα έντονα γράμματα φαίνονται πάλι οι θέσεις του πίνακα. Παρατηρούμε ότι οι δείκτες και των γραμμών και των στηλών αριθμούνται από το 0.

11.1.9 Δήλωση δισδιάστατου πίνακα

Η δήλωση ενός μονοδιάστατου πίνακα γίνεται με την ακόλουθη εντολή:

τύπος όνομα_πίνακα [πλήθος γραμμών][πλήθος στηλών];

Με τη δήλωση αφιερώνεται χώρος μνήμης σε διαδοχικές θέσεις κατάλληλος για να δεχθεί δεδομένα του περιγραφόμενου τύπου και σε πλήθος όσο το γινόμενο των διαφορετικών πληθών των δεικτών. Για παράδειγμα γράφοντας σε ένα πρόγραμμα τη δήλωση ***int***grades[3][5]; ορίζουμε ένα πίνακα με όνομα grades που αποτελείται από 3 γραμμές και 5 στήλες και συνολικά από 15 στοιχεία τύπου ***int***:

grades[0,0]	grades[0,1]	grades[0,2]	grades[0,3]	grades[0,4]
grades[1,0]	grades[1,1]	grades[1,2]	grades[1,3]	grades[1,4]
grades[2,0]	grades[2,1]	grades[2,2]	grades[2,3]	grades[2,4]

Γενικά το στοιχείο ενός δισδιάστατου πίνακα στη γραμμή i και στη στήλη j συμβολίζεται σαν Πίνακας[i,j].

11.1.10Αρχικές τιμές δισδιάστατου πίνακα

Μαζί με την δήλωση ενός δισδιάστατου πίνακα μπορούμε να δώσουμε και αρχικές τιμές στα στοιχεία του πίνακα με εντολή της μορφής:

intgrades[2][5]={100,90,80,70,60,50,40,30,20,10};

Τότε στον πίνακα grades οι ακέραιοι μέσα στις αγκύλες τοποθετούνται κατά γραμμές επομένως θα έχουμε τα στοιχεία:

100	90	80	70	60
50	40	30	20	10

Δηλαδή το grades[0][0] θα είναι ο ακέραιος 100, το grades[1] [2] θα είναι ακέραιος 30 κ.λ.π. Η προηγούμενη δήλωση θα μπορούσε να γίνει και ως εξής:

int grades[2][5]= {{100,90,80,70,60}, {50,40,30,20,10}};

11.1.11Εκχώρηση τιμών στα στοιχεία δισδιάστατου πίνακα

Από τη στιγμή που θα δηλώσουμε ένα πίνακα δεν υπάρχει τρόπος να δώσουμε με μια εντολή νέες τιμές σε όλα τα στοιχεία του (όπως συμβαίνει κατά την δήλωσή του με αρχικές τιμές). Δηλαδή μετά από τη δήλωση: ***int***grades[2][5]; είναι λάθος να γράψουμε:

grades={{100,90,80,70,60,50,40,30,20,10}};

Μπορούμε όμως να δίνουμε τιμές σε μεμονωμένα στοιχεία του πίνακα γράφοντας:

grades[1][3]=100; grades[0][2]=500;

11.1.12Πλήθος Στοιχείων Δισδιάστατου Πίνακα

Ισχύουν τα προαναφερθέντα στους μονοδιάστατους πίνακες. Δηλαδή μετά τη δήλωση:

intgrades[2][5] ={100, 90,80,70,60,50,30,40,20,10};

αν δώσουμε την εντολή:

printf("Count of elements of array grades is %d \n",***sizeof***(grades)/***sizeof***(***int***));

θα λάβουμε την απάντηση 10 όσο και το πλήθος των στοιχείων του πίνακα grades.

Παράδειγμα: Διάβασμα και Εκτύπωση δισδιάστατου πίνακα

Για να διαβάσουμε (γεμίσουμε) ένα δισδιάστατο πίνακα με τιμές και στη συνέχει να τον εκτυπώσουμε στην οθόνη γράφουμε τις ακόλουθες εντολές: (έστω ότι έχει προηγηθεί η δήλωση ***float***grades[3][5];)

```
for (i=0;i<3;i++)
{
    for (j=0;j<5;j++)
    {
        printf("Give value to element of %d row and %s %d column \n",i+1," ",j+1);
        scanf("%d",&grades[i][j]);
    }
}


for (i=0;i<3;i++)
{
   for (j=0;j<5;j++)
   {
       printf("%3d",x[i][j]);
   }
   printf("\n");
}
```

12 Συμβολοσειρές

Μία συμβολοσειρά (string) είναι μια ομάδα χαρακτήρων μέσα σε ένα ζεύγος διπλών εισαγωγικών π.χ. "Holidays", "Programming” κ.λ.π. Σε ένα πρόγραμμα χρησιμοποιούμε συνήθως τις συμβολοσειρές για την εκτύπωση μηνυμάτων στην οθόνη π.χ. η εντολή ***printf***(“Giveanumber”); εκτυπώνει τη φράση Giveanumber στην οθόνη του Η/Υ. Η συμβολοσειρά τοποθετείται στη μνήμη του Η/Υ σαν ένας πίνακας χαρακτήρων με ένα επιπλέον χαρακτήρα στο τέλος του. Αυτός ο χαρακτήρας ονομάζεται μηδενικός (null) και συμβολίζεται ως '\0'. Π.χ. το stringHolidays τοποθετείται στη μνήμη ως εξής:

H	o	l	I	D	a	Y	s	\0

12.1 Ορισμός πίνακα συμβολοσειράς

Μια συμβολοσειρά μπορεί να τοποθετηθεί σε ένα πίνακα τύπου ***char*** δίνοντας αρχική τιμή π.χ. ***char***s[]="onestring"; (προσαρμοζόμενη διάσταση) ή με τη δήλωση ***char***s[11]="onestring"; (δεδομένη διάσταση). Αν και οι χαρακτήρες που τοποθετούνται μέσα στις αποστρόφους είναι 10 και στις 2 προηγούμενες δηλώσεις δημιουργείται ένας μονοδιάστατος πίνακας χαρακτήρων με όνομα s που έχει 11 στοιχεία (στο τέλος του πίνακα τοποθετείται αυτόματα ο μηδενικός χαρακτήρας). Το μέγιστο πλήθος στοιχείων του πίνακα χαρακτήρων στον οποίο θα τοποθετηθεί μια συμβολοσειρά πρέπει να είναι μεγαλύτερο τουλάχιστον κατά ένα από το πλήθος χαρακτήρων που θέλουμε να περιλαμβάνονται στη συμβολοσειρά προκειμένου να τοποθετηθεί στο τέλος του πίνακα ο μηδενικός χαρακτήρας.

12.2 Εκτύπωση συμβολοσειράς με τη συνάρτηση *puts*

Η συνάρτηση ***puts***() που ορίζεται στο αρχείο επικεφαλίδας <***stdio.h***> μπορεί να χρησιμοποιηθεί για να εκτυπώσουμε μια συμβολοσειρά στην οθόνη. Η συνάρτηση δέχεται σαν όρισμα μια συμβολοσειρά και εκτυπώνει στην οθόνη του χαρακτήρες της εκτός του μηδενικού χαρακτήρα. Στο τέλος αποστέλλει και τον χαρακτήρα αλλαγής γραμμής.

12.3 Ανάγνωση συμβολοσειράς από πληκτρολόγιο

Μετά τη δήλωση ενός πίνακα τύπου ***char*** μπορούμε να εισάγουμε μια συμβολοσειρά στον πίνακα από το πληκτρολόγιο με 2 τρόπους: (πάντως και στις 2 περιπτώσεις προστίθεται στο τέλος της συμβολοσειράς ο μηδενικός χαρακτήρας ('\0'))

- Χρησιμοποιώντας την εντολή εισόδου ***scanf*** με όρισμα %s. Στην περίπτωση αυτή εισάγονται οι πληκτρολογούμενοι χαρακτήρες μέχρι να βρεθεί είτε ο χαρακτήρας κενό ή tab ή newline που δημιουργείται από το πλήκτρο enter (αν το string έχει κενά δεν διαβάζεται όλο)
- Χρησιμοποιώντας την συνάρτηση ***gets***(). Η συνάρτηση ορίζεται στο αρχείο <***stdio.h***>.

12.4 Πίνακες με στοιχεία συμβολοσειρές

Ένας μονοδιάστατος πίνακας που τα στοιχεία του είναι συμβολοσειρές στην πραγματικότητα είναι ένας πίνακας 2 διαστάσεων τύπου ***char*** αφού το κάθε στοιχείο του (που είναι μια συμβολοσειρά) είναι ένας μονοδιάστατος πίνακας τύπου ***char***. Έτσι τους μονοδιάστατους πίνακες με συμβολοσειρές τους χειριζόμαστε όπως τους δισδιάστατους πίνακες.

12.5 Δήλωση πίνακα με αρχικές τιμές συμβολοσειρές

Με την δήλωση ***char***s[ ][15] = {"ΤΕΙ Πειραιά", "ΤΕΙ Αθηνών", "ΤΕΙ Χαλκίδας"}; δημιουργείται ο ακόλουθος δισδιάστατος πίνακας s που έχει μορφή:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
0	Τ	Ε	Ι		Π	Ε	Ι	ρ	α	Ι	ά	\0			
1	Τ	Ε	Ι		Α	Θ	Η	ν	ώ	ν	\0				
2	Τ	Ε	Ι		Χ	Α	λ	κ	ί	δ	α	ς	\0		

Τα στοιχεία του πίνακα μπορούμε να τα προσπελάσουμε με δύο δείκτες που συμβολίζουν τη γραμμή και τη στήλη αντίστοιχα π.χ. το στοιχείο s[0][1] είναι ο χαρακτήρας 'Ε', το στοιχείο s[2][11] είναι ο χαρακτήρας 'ς' κ.λ.π.

12.6 Συναρτήσεις επεξεργασίας συμβολοσειρών

Για την επεξεργασία των συμβολοσειρών χρησιμοποιούμε τις ακόλουθες συναρτήσεις οι οποίες ορίζονται στο αρχείο επικεφαλίδας <***string.h***>.

12.6.1 Υπολογισμός μήκους συμβολοσειράς

Για να υπολογίσουμε το πλήθος των χαρακτήρων μιας συμβολοσειράς χρησιμοποιούμε τη συνάρτηση:

strlen(συμβολοσειρά)

Παράδειγμα

chars[15]= "ΤΕΙ ΠΑΤΡΑΣ";

Γράφοντας την εντολή:

printf("%d \n", ***strlen***(s));

τυπώνεται η τιμή 10 (όσοι και οι χαρακτήρες της συμβολοσειράς).

Παρατηρήσεις

1. Η συνάρτηση ***strlen*** επιστρέφει το μήκος μιας συμβολοσειράς δηλαδή το πλήθος των χαρακτήρων της χωρίς όμως το μηδενικό χαρακτήρα.
2. Η συνάρτηση ***strlen*** δεν τροποποιεί τη συμβολοσειρά π.χ. γράφοντας την εντολή ***printf***("%d\n", ***strlen***("COMPUTER"); τυπώνεται η τιμή 8.
3. Ο υπολογισμός του μήκους μιας συμβολοσειράς εκτός από τη συνάρτηση ***strlen*** μπορεί να γίνει και με τη μέτρηση των χαρακτήρων της συμβολοσειράς μέχρι το μηδενικό της χαρακτήρα.

12.6.2 Αντιγραφή συμβολοσειράς

Για να αντιγράψουμε μια συμβολοσειρά2 σε μια συμβολοσειρά1 χρησιμοποιούμε τη συνάρτηση:

strcpy(συμβολοσειρά1, συμβολοσειρά2)

Με τη συνάρτηση αυτή αντιγράφεται και ο μηδενικός χαρακτήρας τερματισμού της συμβολοσειράς.

Παράδειγμα

chars[10], t[10]=”ATEI”;

Η εντολή ***strcpy***(s,t) αντιγράφει τη συμβολοσειρά t στη συμβολοσειρά s δηλαδή η συμβολοσειρά s λαμβάνει σαν περιεχόμενο τη λέξη ATEI. Συνεπώς η εντολή:

printf(“%s \t %s \n”, s ,t)

θα τυπώσει 2 φορές τη λέξη ATEI αφού και οι 2 συμβολοσειρές έχουν το ίδιο περιεχόμενο.

Παρατήρηση

Αν κατά την εκτέλεση της εντολής ***strcpy***(s,t) η συμβολοσειρά s έχει περιεχόμενο τότε αυτό επικαλύπτεται από το περιεχόμενο της συμβολοσειράς t.

12.6.3 Προσάρτηση συμβολοσειράς

Για να προσαρτήσουμε μια συμβολοσειρά2 στο τέλος μιας συμβολοσειράς1 χρησιμοποιούμε τη συνάρτηση:

strcat(συμβολοσειρά1, συμβολοσειρά2)

Για τη συμβολοσειρά1 πρέπει να έχει δηλωθεί αρκετός χώρος μνήμης ώστε να τοποθετηθούν οι προσαρτώμενοι χαρακτήρες χωρίς να επεκταθούν πέρα από το χαρακτήρα τερματισμού της συμβολοσειράς.

Παράδειγμα

char s[20]=”Computer”, t[10]=”Magazine”;

Η εντολή ***strcat***(s,t) συνενώνει (προσαρτεί) στο τέλος της συμβολοσειράς s τη συμβολοσειρά t δηλαδή τη λέξη Magazine. Συνεπώς η εντολή:

printf(“%s \t %s \n”, s , t);

θα τυπώσει για τη συμβολοσειρά s τη φράση ComputerMagazine και για τη συμβολοσειρά t τη λέξη Magazine.

12.6.4 Σύγκριση συμβολοσειρών

Για να συγκρίνουμε 2 συμβολοσειρές χρησιμοποιούμε τη συνάρτηση:

ακέραια μεταβλητή= ***strcmp***(συμβολοσειρά1, συμβολοσειρά2)

Η συνάρτηση ***strcmp*** συγκρίνει τις 2 συμβολοσειρές χαρακτήρα προς χαρακτήρα μέχρι να βρεθεί ανόμοιος χαρακτήρας ή το τέλος μιας από τις 2 συμβολοσειρές. Συγκρίνεται πάντα η πρώτη συμβολοσειρά σε σχέση με τη δεύτερη και στην ακέραια μεταβλητή επιστρέφεται κάποιο από τα ακόλουθα αποτελέσματα:

- Θετικός αριθμός αν η συμβολοσειρά1>συμβολοσειρά2

- Μηδέν αν οι 2 συμβολοσειρές είναι ίδιες
- Αρνητικός αριθμός αν η συμβολοσειρά1<συμβολοσειρά2

Παράδειγμα
char s[20]=”Computer”, t[10]=” Magazine”;
int t;
Η εντολή t= ***strcmp***(s,t) επιστρέφει στη μεταβλητή t θετικό αριθμό γιατί η λέξη Computer προηγείται (αλφαβητικά) από τη λέξη Magazine.

12.7 Συναρτήσεις ελέγχου χαρακτήρων

Οι συναρτήσεις ελέγχου χαρακτήρων εξετάζουν το είδος των χαρακτήρων που εισάγονται και ορίζονται στο αρχείο επικεφαλίδας <***stdlib.h***>. Οι πιο βασικές είναι οι ακόλουθες:

Πρωτότυπο συνάρτησης	Επιστρεφόμενη τιμή
int isalnum(***int*** c);	Διάφορη του 0 αν το c είναι αλφαριθμητικό διαφορετικά επιστρέφεται 0
int isalpha(***int*** c);	Διάφορη του 0 αν το c είναι αλφαβητικός διαφορετικά επιστρέφεται 0
int isdigit(***int*** c);	Διάφορη του 0 αν το c είναι ψηφίο διαφορετικά επιστρέφεται 0
int isspace(***int*** c);	Διάφορη του 0 αν το c είναι κενό διάστημα ή tab διαφορετικά επιστρέφεται 0
int ispunct(***int*** c);	Διάφορη του 0 αν το c είναι σημείο στίξης διαφορετικά επιστρέφεται 0
int islower(***int*** c);	Διάφορη του 0 αν το c είναι πεζό γράμμα διαφορετικά επιστρέφεται 0
int isupper(***int*** c);	Διάφορη του 0 αν το c είναι κεφαλαίο γράμμα διαφορετικά επιστρέφεται 0

Παρατηρήσεις

1. Αλφαριθμητικοί χαρακτήρες ονομάζονται αυτοί που αντιπροσωπεύουν γράμμα ή ψηφίο.
2. Αλφαβητικοί χαρακτήρες ονομάζονται αυτοί που αντιπροσωπεύουν μόνο γράμμα.
3. Τα σημεία στίξης είναι π.χ. οι χαρακτήρες ! . , : ‘ ? + - * / { } [ ] <> = # $ κ.λ.π.

12.8 Συναρτήσεις μετατροπής χαρακτήρων

Οι συναρτήσεις μετατροπής χαρακτήρων αλλάζουν τον τρόπο γραφής τα ων χαρακτήρων και ορίζονται στο αρχείο επικεφαλίδας <***ctype.h***>. Είναι οι ακόλουθες:

Πρωτότυπο Συνάρτησης	Επιστρεφόμενη τιμή
int tolower(***int*** c);	Ο χαρακτήρας c μετατρέπεται σε πεζό γράμμα
int toupper(***int*** c);	Ο χαρακτήρας c μετατρέπεται σε κεφαλαίο γράμμα

Παρατήρηση
Οι συναρτήσεις tolower και toupper δεν επηρεάζουν χαρακτήρα που δεν αντιπροσωπεύει γράμμα.

13 Δείκτες

Η μεταβλητή ενός δείκτη (pointer) έχει σαν περιεχόμενο τη διεύθυνση μνήμης (σε δεκαεξαδική μορφή) μιας άλλης μεταβλητής (αυτής στην οποία δείχνει).

13.1 Δήλωση μεταβλητής δείκτη

Για να δηλώσουμε μια μεταβλητή δείκτη γράφουμε την ακόλουθη εντολή:

τύπος *όνομα_μεταβλητής;

π.χ. γράφοντας την εντολή ***int*** *x; δηλαδή η x είναι δείκτης (μεταβλητή δείκτης) που δείχνει σε ακέραιο (δηλαδή σε μια θέση μνήμης με ακέραιο περιεχόμενο). Σχηματικά έχουμε x **à** ***int***

π.χ. γράφοντας την εντολή ***float*** *y; δηλαδή η y είναι δείκτης (μεταβλητή δείκτης) που δείχνει σε πραγματικό (δηλαδή σε μια θέση μνήμης με πραγματικό περιεχόμενο). Σχηματικά έχουμε y **à** ***float***

π.χ. γράφοντας την εντολή ***char*** *z; δηλαδή η z είναι δείκτης (μεταβλητή δείκτης) που δείχνει σε χαρακτήρα (δηλαδή σε μια θέση μνήμης με περιεχόμενο χαρακτήρα). Σχηματικά έχουμε z **à** ***char***

13.2 Απόδοση τιμής σε δείκτη

Για να καταχωρίσουμε τιμή σε μια μεταβλητή δείκτη γράφουμε την εντολή:

μεταβλητή_δείκτης= &μεταβλητή που δείχνει

Το σύμβολο & υποδηλώνει διεύθυνση.

Για παράδειγμα αν έχουμε τις δηλώσεις ***int***a=5, *x; τότε με την εντολή x=&a; καταχωρούμε στο δείκτη x ως περιεχόμενο τη διεύθυνση της μεταβλητής a. Σχηματικά έχουμε:

Επίσης αν έχουμε τις δηλώσεις ***float***b=3.5, *y; τότε με την εντολή y=&b; καταχωρούμε στο δείκτη y ως περιεχόμενο τη διεύθυνση της μεταβλητής b. Σχηματικά έχουμε:

13.3 Προσπέλαση μεταβλητής στην οποία δείχνει ο δείκτης

Για να προσπελάσουμε το περιεχόμενο της μεταβλητής στην οποία δείχνει ένας δείκτης χρησιμοποιούμε την εντολή:

*μεταβλητή_δείκτης

Για παράδειγμα αν έχουμε τις δηλώσεις ***int***a=5, *x; τότε με την εντολή x=&a; καταχωρούμε στο δείκτη x ως περιεχόμενο τη διεύθυνση της μεταβλητής a και με την εντολή ***printf***("%d\n",*x); τυπώνεται η τιμή 5.

Επίσης αν έχουμε τις δηλώσεις ***float***b=3.5, *y; τότε με την εντολή y=&b; καταχωρούμε στο δείκτη y ως περιεχόμενο τη διεύθυνση της μεταβλητής b και με την εντολή ***printf***("%f\n",*y); τυπώνεται η τιμή 3.5.

Παράδειγμα

inta=2, *x;
x=&a;
printf("%d %d \n",a, *x); **à** τυπώνονται οι τιμές 2 και 2 αντίστοιχα.

*x=4;

printf("%d %d \n",a, *x); τυπώνονται οι τιμές 4 και 4 αντίστοιχα. Με την εντολή *x=4 δίνουμε την τιμή 4 στο a δηλαδή ***x=4 ίδιο a=4**. Με τους δείκτες κάνουμε δηλαδή έμμεση αναφορά σε μεταβλητές.

13.4 Δείκτες και μονοδιάστατοι πίνακες

Στη C υπάρχει άμεση σχέση ανάμεσα στους δείκτες και στους μονοδιάστατους πίνακες. Συγκεκριμένα το όνομα ενός μονοδιάστατου πίνακα ταυτίζεται με τη διεύθυνση του αρχικού (μηδενικού) στοιχείου του πίνακα. Για παράδειγμα έστω ότι έχουμε δηλώσει τον ακόλουθο πίνακα:

intx[10];.

Ισχύει ότι:

x^{o} &x[0]

Ο συμβολισμός x+i μας δίνει τη διεύθυνση του i-οστου στοιχείου του πίνακα. Άρα ισοδύναμα μπορούμε να γράψουμε:

x[i]=*(x+i)

Παράδειγμα

```
#include<stdio.h>
voidmain()
{
int i, x[10];

   for (i=0;i<10;i++)
   {
        printf("Give %d value: ",i+1);
        scanf("%d",&(*(x+i)));
   }

   for (i=0;i<10;i++)
   {
        printf("%5d",*(x+i));
   }
   printf("\n");
}
```

13.5 Δείκτες και δισδιάστατοι πίνακες

Στη C υπάρχει επίσης άμεση σχέση ανάμεσα στους δείκτες και στους δισδιάστατους πίνακες. Συγκεκριμένα έστω ότι έχουμε δηλώσει και αρχικοποιήσει τον ακόλουθο πίνακα:

int r[2][5]={100,90,80,70,60,50,40,30,20,10};

Ισχύειότι:

r[i][j]=*(r[i]+j]) = *(r+i)[j] = *(*(r+i)+j)

Αυτό φαίνεται σχηματικά και ως εξής:

δείκτης r	**πίνακας διευθύνσεων**	**πίνακας στοιχείων**				
διεύθυνση **à**	διεύθυνση r[0] **à**	100	90	80	70	60
	διεύθυνση r[1] **à**	50	40	30	20	10

Παράδειγμα

```
#include<stdio.h>
voidmain()
```

```
{
int i, j, r[2][5]={100,90,80,70,60,50,40,30,20,10};

    for (i=0;i<2;i++)
          {
             for (j=0;j<5;j++)
                   printf("%d",*(*(r+i)+j));
             printf("\n");
          }
```

Σημείωση

Η διεύθυνση του πρώτου στοιχείου του πίνακα μπορεί να εκφραστεί ως *r ή &r[0][0]

14 Υποπρογράμματα (συναρτήσεις)

14.1 Βασικές έννοιες

Η C είναι μια γλώσσα προγραμματισμού της 3ης γενιάς των γλωσσών προγραμματισμού και δημιουργήθηκε για την ανάπτυξη λειτουργικών συστημάτων και είναι κατάλληλη γα την κατασκευή κάθε είδους λογισμικού. Π.χ. σε C έχει γραφεί το λειτουργικό σύστημα Unix.

Σαν είδος προγραμματισμού η C ανήκει στην κατηγορία του δομημένου προγραμματισμού. Τα βασικά χαρακτηριστικά του δομημένου προγραμματισμού είναι τα ακόλουθα:

1. Ένα πρόγραμμα δεν υλοποιείται σαν μια ενιαία οντότητα αλλά διασπάται σε ανεξάρτητα υποπρογράμματα τα οποία στη C ονομάζονται συναρτήσεις
2. Το καθένα από αυτά τα υποπρογράμματα υλοποιεί μια διαφορετική λειτουργία του προγράμματος
3. Τα διάφορα υποπρογράμματα είναι μεταξύ τους ιεραρχικά δομημένα. Αυτό σημαίνει ότι υπάρχει ένα κύριο πρόγραμμα που στη C ορίζεται από τη συνάρτηση ***main***() το οποίο καλεί όλα τα υπόλοιπα υποπρογράμματα

Συμπερασματικά μπορούμε να χαρακτηρίσουμε ένα δομημένο πρόγραμμα ως τμηματικό πρόγραμμα αφού αποτελείται από αυτόνομες μονάδες που συνδέονται μεταξύ τους.

Τα πλεονεκτήματα του δομημένου προγραμματισμού είναι τα ακόλουθα:

1. Μας παρέχει ευκολία στην ανάπτυξη μεγάλων προγραμμάτων γιατί ο προγραμματιστής προκειμένου να υλοποιήσει ένα μεγάλο πρόγραμμα το αναλύει σε απλούστερα και μικρότερα υποπρογράμματα (συναρτήσεις) τα οποία είναι πιο εύκολο να αναπτύξει από ότι όλο το πρόγραμμα. Στη συνέχεια απλώς καλεί όλα αυτά τα υποπρογράμματα μέσα από το κύριο πρόγραμμα (συνάρτηση ***main***()).
2. Μας παρέχει ταχύτητα στην ανάπτυξη προγραμμάτων γιατί ένα μεγάλο πρόγραμμα το οποίο θα αποτελείται από ένα πλήθος υποπρογραμμάτων μπορεί να αναπτυχθεί από μια ομάδα προγραμματιστών όπου το κάθε μέλος αναλαμβάνει να γράψει κάποια υποπρογράμματα ταυτόχρονα με τα υπόλοιπα μέλη με αποτέλεσμα ο συνολικός χρόνος ανάπτυξης του προγράμματος να μειώνεται σημαντικά.
3. Επιταχύνεται η διαδικασία ελέγχου και εκσφαλμάτωσης του προγράμματος αφού κάθε υποπρόγραμμα μπορεί να ελεγχθεί ξεχωριστά και ανεξάρτητα από τα υπόλοιπα υποπρογράμματα
4. Μας παρέχει ευελιξία στην ανάπτυξη προγραμμάτων γιατί αν κάποιο τμήμα προγράμματος που κάνει μια συγκεκριμένη δουλειά πρέπει να επαναλαμβάνεται σε πολλά σημεία ενός προγράμματος τότε μπορεί να δηλωθεί ως υποπρόγραμμα και απλά να καλείται σε όσα σημεία απαιτείται. Επίσης δίνεται η δυνατότητα στους προγραμματιστές να αναπτύσσουν υποπρογράμματα γενικής χρήσης τα οποία μπορούν να ενσωματώσουν σε κάθε πρόγραμμα που γράφουν μειώνοντας με αυτό τον τρόπο τον απαιτούμενο προγραμματιστικό χρόνο και κόπο
5. Μας παρέχει ευκολία συντήρησης στα προγράμματα που αναπτύσσουμε γιατί αν απαιτηθούν αλλαγές σε ένα πρόγραμμα αυτές μπορούν να απομονωθούν σε επίπεδο υποπρογραμμάτων χωρίς να αλλάξει ολόκληρη η δομή του προγράμματος

14.2 .Ορισμός συνάρτησης

Ο ορισμός συνάρτησης αποτελείται από δυο τμήματα: την επικεφαλίδα και την ομάδα εντολών μέσα σε άγκιστρα του σώματος εντολών. Ο ορισμός μιας συνάρτησης έχει την ακόλουθη μορφή:

Ένα υποπρόγραμμα (συνάρτηση) γράφεται με τον ακόλουθο τρόπο:

τύπος επιστρεφόμενης τιμής ή ***void***όνομα_συνάρτησης (λίστα παραμέτρων ή κενό)
{
ομάδα εντολών;
return επιστρεφόμενη τιμή; ή τίποτα αν έχουμε δηλώσει ***void***

}

Από τον τρόπο γραφής παρατηρούμε ότι ένα υποπρόγραμμα όταν τελειώνει μπορεί είτε να επιστρέφει στο κύριο πρόγραμμα που το κάλεσε ένα αποτέλεσμα (οπότε δηλώνεται ο τύπος του αποτελέσματος που επιστρέφεται) είτε να μην επιστρέφει κανένα αποτέλεσμα οπότε δηλώνεται η λέξη ***void***. Μετά γράφουμε το όνομα του υποπρογράμματος το οποίο επιλέγεται από εμάς και αφενός πρέπει να είναι αντιπροσωπευτικό της λειτουργίας του υποπρογράμματος και αφετέρου να είναι μια συνεχόμενη λέξη χωρίς κενά διαστήματα. Οι παράμετροι είναι τοπικές μεταβλητές που ορίζονται στην επικεφαλίδα του υποπρογράμματος και χρησιμοποιούνται για την επικοινωνία με το κύριο πρόγραμμα. Συγκεκριμένα οι παράμετροι δέχονται δεδομένα από το κύριο πρόγραμμα κατά την κλήση του υποπρογράμματος προκειμένου αυτό να τα χρησιμοποιήσει για τον υπολογισμό αποτελεσμάτων. Η χρήση των παραμέτρων είναι προαιρετική δηλαδή αν το υποπρόγραμμα δεν θέλουμε να λαμβάνει δεδομένα από το κύριο πρόγραμμα τότε παραλείπονται.

Όταν θέλουμε ένα υποπρόγραμμα τη στιγμή που τελειώνει να επιστρέφει αποτέλεσμα στο κύριο πρόγραμμα τότε χρησιμοποιούμε την εντολή ***return*** αποτέλεσμα; όπου το αποτέλεσμα είναι είτε μια σταθερά είτε μια μεταβλητή π.χ.

return 1;
ή
returnx;

Στην περίπτωση που το υποπρόγραμμα επιστρέφει τιμή πρέπει όπως αναφέραμε να δηλώσουμε στην επικεφαλίδα του υποπρογράμματος τον τύπο του επιστρεφόμενου αποτελέσματος.

14.3 Θέση γραφής υποπρογραμμάτων

Τα υποπρογράμματα μπορούν να γραφούν σε μια από τις ακόλουθες θέσεις:

α)Πριν από το κύριο πρόγραμμα δηλαδή πριν από τη συνάρτηση *main*() όπως φαίνεται ακολούθως:

τύπος αποτελέσματος ή ***void***function1(παράμετροι ή κενό)
{
ομάδα εντολών;
}

τύπος αποτελέσματος ή ***void***function2(παράμετροι ή κενό)
{
ομάδα εντολών;
}
…..

voidmain()
{
κλήση function1;

κλήση function2;
}

β)Μετά από το κύριο πρόγραμμα δηλαδή μετά από τη συνάρτηση *main*()όπως φαίνεται ακολούθως:

δήλωση πρωτοτύπου function1
δήλωση πρωτοτύπου function2

***void*main**()
{
    κλήση function1;
    κλήση function2;
}

τύπος αποτελέσματος ή ***void***function1(παράμετροι ή κενό)
{
    ομάδα εντολών;
}

τύπος αποτελέσματος ή ***void***function2(παράμετροι ή κενό)
{
    ομάδα εντολών;
}
…..

Στην περίπτωση αυτή πρέπει υποχρεωτικά να δηλώσουμε πριν τη συνάρτηση ***main***() το πρωτότυπο όλων των υποπρογραμμάτων (συναρτήσεων) που πρόκειται να καλέσει η ***main***(). Αν δεν γίνει αυτό τότε ο compiler θα θεωρήσει τα ονόματα των συναρτήσεων function1 και function2 ως άγνωστες μεταβλητές. Αναλυτικές πληροφορίες για το πρωτότυπο συνάρτησης δίνονται στην ακόλουθη παράγραφο.

14.4 Πρωτότυπο συνάρτησης

Το πρωτότυπο (prototype) μιας συνάρτησης είναι η πληροφορία που θα πρέπει να δοθεί στον compiler (εφόσον η συνάρτηση καλείται πριν από τον ορισμός της) σχετικά με τον τύπο της συνάρτησης και τις τυπικές παραμέτρους της. Το πρωτότυπο μιας συνάρτησης είναι μια δήλωση της μορφής:

τύπος_τιμής_επιστροφής όνομα_συνάρτησης (λίστα παραμέτρων);

Δηλαδή στο πρωτότυπο μιας συνάρτησης αναφέρουμε την επικεφαλίδα και μόνο που χρησιμοποιούμε κατά τον ορισμό της συνάρτησης. Είδαμε (στον ορισμό της συνάρτησης) ότι στην λίστα παραμέτρων αναφέρουμε τον τύπο και ένα τυπικό (υποθετικό) όνομα για κάθε παράμετρο. Στο πρωτότυπο της συνάρτησης (και μόνο) επιτρέπεται να παραλείψουμε το τυπικό όνομα και να αναφέρουμε μόνο τον τύπο της παραμέτρου. Στο πρόγραμμα πριν χρησιμοποιήσουμε συνάρτηση της βιβλιοθήκης θα πρέπει να δηλώσουμε το πρωτότυπό της. Η δήλωση των πρωτοτύπων γίνεται με τα αρχεία επικεφαλίδων (header ή ***include***files) τα οποία καλούμε να ενσωματωθούν στον κώδικα του προγράμματός μας, χρησιμοποιώντας την δια-

ταγή #***include***. Στα αρχεία επικεφαλίδων υπάρχουν οι δηλώσεις των πρωτότυπων των συναρτήσεων βιβλιοθήκης, που θα χρησιμοποιήσουμε.

Για να επανέλθουμε στο παράδειγμα μας με τη δήλωση του πρωτοτύπου των συναρτήσεων function1 και function2 ουσιαστικά γνωστοποιούμε στον compiler ότι τα ονόματα function1 και function2 είναι ονόματα υποπρογραμμάτων ώστε να μην προκληθεί συντακτικό λάθος κατά την κλήση τους. Όταν όμως τα υποπρογράμματα προηγούνται του ***main***() τότε ή δήλωση του πρωτοτύπου των συναρτήσεων δεν χρειάζεται γιατί ο compiler γνωρίζει τα υποπρογράμματα αφού έχει γραφεί ο κώδικας τους πριν την κλήση τους.

14.5 Τοπικές και σφαιρικές μεταβλητές

Κάθε μεταβλητή που ορίζεται μέσα σε μια συνάρτηση είναι τοπική (local) για την συνάρτηση. Αυτό σημαίνει ότι παύει να υφίσταται μετά την εκτέλεση των εντολών της συνάρτησης. Επίσης οι παράμετροι μιας συνάρτησης (εφόσον υπάρχουν) είναι επίσης τοπικές μεταβλητές. Κάθε μεταβλητή που ορίζεται εκτός από κάθε συνάρτηση (στην αρχή ενός υποπρογράμματος) ονομάζεται καθολική ή σφαιρική μεταβλητή (global). Αν υπάρχει καθολική μεταβλητή (οριζόμενη εκτός συνάρτησης) με το ίδιο όνομα η τοπική μεταβλητή αποκρύπτει την καθολική κατά την εκτέλεση της συνάρτησης.

Παράδειγμα 1

```
#include<stdio.h>
int k=1;

int func1(int x, int y)
{
    int sum;

    sum= x+y;
    k++;
    return sum;
}

void func2(int x, int y)
{
    int dif;

    dif= x-y;
    k++;
    printf("The difference is %d \n", dif);
}

voidmain()
{
     int a=1,b=0;

     printf("The sum is %d \n",func1(a,b);
     func2(a,b);
     printf("%d \n", k);
}
```

Οι μεταβλητές a, b είναι τοπικές για τη συνάρτηση ***main***(). Αυτό σημαίνει ότι υπάρχουν μόνο μέσα στη ***main***(). Έξω από τη συνάρτηση αυτή δεν μπορούν να χρησιμοποιηθούν γιατί απλά δεν υφίστανται. Ομοίως η μεταβλητή sum είναι τοπική για τη συνάρτηση func1 και η dif είναι τοπική για τη συνάρτηση func2. Ομοίως οι παράμετροι x, y είναι τοπικές μεταβλητές για τις συναρτήσεις func1 και func2. Είναι πολύ σημαντικό να τονίσουμε ότι οι μεταβλητές x, y της συνάρτησης func1 είναι διαφορετικές από τις μεταβλητές x, y της συνάρτησης func2 έστω και αν έχουν το ίδιο όνομα. Η μεταβλητή k είναι καθολική σε όλο το πρόγραμμα επειδή δηλώνεται στην αρχή πριν από όλες τις συναρτήσεις. Αυτό σημαίνει ότι η μεταβλητή k μπορεί να χρησιμοποιηθεί σε όλα τα υποπρογράμματα και είναι η ίδια μεταβλητή παντού. Όταν συνεπώς εκτυπώσουμε τη μεταβλητή k στο κύριο πρόγραμμα (στη συνάρτηση ***main***()) με την εντολή ***printf***("%d \n", k); η τιμή της k είναι 3 αφού όπως παρατηρούμε με τη δήλωση της η μεταβλητή k παίρνει την τιμή 1, στη συνέχεια μέσα στη συνάρτηση func1 αυξάνεται κατά 1 και γίνεται 2, μετά στη συνάρτηση func2 αυξάνεται επίσης κατά 1 και γίνεται 3 και τελικά όταν εκτυπώνεται στο ***main*** έχει την τιμή 3.

14.6 Τρόποι κλήσης συνάρτησης με παραμέτρους

Η κλήση μιας συνάρτησης (με παραμέτρους) από το κύριο πρόγραμμα μπορεί να γίνει με τους ακόλουθους τρόπους:

14.6.1 Πέρασμα Τιμής

Ο πρώτος τρόπος κλήσης μιας συνάρτησης ονομάζεται πέρασμα τιμής (callbyvalue). Στην περίπτωση αυτή διοχετεύονται προς τις παραμέτρους της καλούμενης συνάρτησης τιμές σταθερών ή μεταβλητών από το κύριο πρόγραμμα.

Παράδειγμα

```
#include<stdio.h>

int func1(int x, int y)
{
int sum;

   sum= x+y;
   return sum;
}

void func2(int x, int y)
{
int dif;

   dif= x-y;
   printf("The difference is %d \n", dif);
}

voidmain()
{
int a=1,b=0;

   printf("The sum is %d \n",func1(a,b);
   func2(4, 5);
```

}

Η κλήση της συνάρτησης func1 από το κύριο πρόγραμμα γίνεται με την εντολή

printf(“The sum is %d \n”,func1(a,b);

Κατά την κλήση αυτή οι τιμές των μεταβλητών a, b καταχωρούνται στις (τοπικές) μεταβλητές x και y αντίστοιχα (της συνάρτησης func1). Κατά την κλήση της func1 γίνονται τα εξής:

Μεταβλητές κύριου προγράμματος	**Παράμετροι func 1**
a **à**	x
b **à**	y

Αυτός ο τρόπος κλήσης της συνάρτησης func1 ονομάζεται πέρασμα με τιμή διότι περνάμε τιμές στις παραμέτρους της func1.

Η κλήση της συνάρτησης func2 από το κύριο πρόγραμμα γίνεται με την εντολή

func2(4, 5);

Κατά την κλήση αυτή οι τιμές 4, 5 καταχωρούνται στις (τοπικές) μεταβλητές x και y αντίστοιχα (της συνάρτησης func2). Κατά την κλήση της func2 γίνονται τα εξής:

Μεταβλητές κύριου προγράμματος	**Παράμετροι func 2**
4 **à**	x
5 **à**	y

Αυτός ο τρόπος κλήσης της συνάρτησης func1 ονομάζεται επίσης πέρασμα με τιμή διότι περνάμε (σταθερές) τιμές στις παραμέτρους της func2.

14.6.2 Πέρασμα αναφοράς

Ο δεύτερος τρόπος κλήσης μιας συνάρτησης ονομάζεται πέρασμα αναφοράς (call-byreference). Στην περίπτωση αυτή διοχετεύονται προς τις παραμέτρους της καλούμενης συνάρτησης οι διευθύνσεις μεταβλητών από το κύριο πρόγραμμα.

Παράδειγμα1

```
#include<stdio.h>

void func1(int x, int y, int *sum, int *dif)
{

   *sum=x+y;
   *dif= x-y;
}

voidmain()
{
int a=1,b=2, res1, res2;

   func1(a, b, &res1, &res2);
   printf(“The sum is %d \n”, res1);
   printf(“The difference is %d \n”, res2);
}
```

Η κλήση της συνάρτησης func1 από το κύριο πρόγραμμα γίνεται με την εντολή

func1(a, b, &res1, &res2);

Κατά την κλήση αυτή οι τιμές των μεταβλητών a, b καταχωρούνται στις (τοπικές) μεταβλητές x και y αντίστοιχα (της συνάρτησης func1) δηλαδή έχουμε πέρασμα με τιμή, ενώ οι διευθύνσεις των μεταβλητών res1 και res2 καταχωρούνται στις (τοπικές) μεταβλητές δείκτη sum και dif της func1 αντίστοιχα δηλαδή έχουμε πέρασμα με αναφορά. Κατά την κλήση της func1 γίνονται τα εξής:

Μεταβλητές κύριου προγράμματος	**Παράμετροι func 1**
a **à**	x
b **à**	y
&res1 **à**	sum
&res2 **à**	dif

Με την εντολή *sum= x+y; καταχωρούμε στο περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης sum την τιμή 1+2=3. Η μεταβλητή που δείχνει ο δείκτης sum είναι η res1, συνεπώς στο περιεχόμενο της res1 του κύριου προγράμματος θέτουμε την τιμή 3 μέσα από το υποπρόγραμμα func1. Άρα χρησιμοποιώντας δείκτες σαν παραμέτρους μιας συνάρτησης μπορούμε να προσπελάσουμε απευθείας μέσα από το υποπρόγραμμα μεταβλητές του κύριου προγράμματος. Ομοίως με την εντολή *dif=x-y; καταχωρούμε στο περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης dif την τιμή 1-2=-1. Η μεταβλητή που δείχνει ο δείκτης dif είναι η res2, συνεπώς στο περιεχόμενο της res2 του κύριου προγράμματος θέτουμε την τιμή -1 μέσα από το υποπρόγραμμα func1. Συνεπώςοιεντολές:

printf(“The sum is %d \n”, res1);

printf(“The difference is %d \n”, res2);

στο κύριο πρόγραμμα θα τυπώσουν τις τιμές 3 και –1 για τις μεταβλητές res1 και res2 αντίστοιχα.

Το πέρασμα αναφοράς χρησιμοποιείται συνήθως όταν θέλουμε ένα υποπρόγραμμα να επιστρέφει περισσότερα από ένα αποτελέσματα στο κύριο πρόγραμμα. Δηλώνοντας τον τύπο του αποτελέσματος που επιστρέφεται από ένα υποπρόγραμμα και χρησιμοποιώντας την εντολή ***return*** για την επιστροφή του αποτελέσματος επιστρέφεται πάντα μόνο 1 τιμή από το υποπρόγραμμα στο κύριο πρόγραμμα, ενώ χρησιμοποιώντας δείκτες σαν παραμέτρους (πέρασμα με αναφορά) μπορούμε όπως αναφέραμε να επιστρέφουμε περισσότερες από 1 τιμές στο κύριο πρόγραμμα αφού μπορούμε έμμεσα να προσπελάσουμε τις μεταβλητές του κύριου προγράμματος.

Παράδειγμα2

```
#include<stdio.h>

void swap(int *x, int *y)
{
int temp;

   temp= *x;
   *x= *y;
   *y= temp;
}

voidmain()
{
int a=1,b=2;

   printf(%d %d \n”, a, b);
```

```
    swap(&a, &b);
    printf(%d %d \n", a, b);
}
```

Η κλήση της συνάρτησης swap από το κύριο πρόγραμμα γίνεται με την εντολή

swap(&a, &b);

Κατά την κλήση αυτή οι διευθύνσεις των μεταβλητών a, b καταχωρούνται στις (τοπικές) μεταβλητές δείκτη x και y αντίστοιχα (της συνάρτησης swap) δηλαδή έχουμε πέρασμα με αναφορά. Κατά την κλήση της swap γίνονται τα εξής:

Μεταβλητές κύριου προγράμματος	**Παράμετροι func 1**
&a **à**	x
&b **à**	y

Με την εντολή temp= *x; καταχωρούμε στη μεταβλητή temp το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης x (δηλαδή το περιεχόμενο της μεταβλητής a του κύριου προγράμματος) άρα η temp παίρνει την τιμή 1.

Με την εντολή *x= *y; καταχωρούμε στη μεταβλητή στην οποία δείχνει ο δείκτης x (δηλαδή στη μεταβλητή a του κύριου προγράμματος) το περιεχόμενο της μεταβλητής στην οποία δείχνει ο δείκτης y (δηλαδή το περιεχόμενο της μεταβλητής b του κύριου προγράμματος) άρα η μεταβλητή a παίρνει την τιμή 2.

Με την εντολή *y= temp; καταχωρούμε στη μεταβλητή στην οποία δείχνει ο δείκτης y (δηλαδή στη μεταβλητή b του κύριου προγράμματος) το περιεχόμενο της μεταβλητής temp άρα η μεταβλητή b παίρνει την τιμή 1.

Συνεπώς όταν στο κύριο πρόγραμμα εκτυπώνουμε αρχικά τις τιμές των μεταβλητών a και b (πριν από την κλήση της συνάρτησης swap) οι τιμές αυτές είναι a=1 και b=2, ενώ όταν εκτυπώνουμε τελικά τις τιμές των μεταβλητών a και b (μετά την κλήση της συνάρτησης swap) οι τιμές αυτές είναι a=2 και b=1 δηλαδή έχουν εναλλαχθεί.

Το πέρασμα αναφοράς χρησιμοποιείται και πάλι προκειμένου να εναλλάξουμε τις τιμές 2 μεταβλητών του κύριου προγράμματος.

14.7 Περιορισμοί κατά την κλήση συνάρτησης με παραμέτρους

Όταν καλούμε μια συνάρτηση με παραμέτρους (είτε με πέρασμα τιμής είτε με πέρασμα αναφοράς) πρέπει να ικανοποιούνται απαραίτητα οι ακόλουθοι 2 περιορισμοί:

Α! Περιορισμός

Ο αριθμός των παραμέτρων που γράφεται στην κλήση της συνάρτησης πρέπει να είναι ίδιος με τον αριθμό των παραμέτρων που γράφεται στον ορισμό της συνάρτησης.

Παραδείγματα

```
void func1(int x, int y)
{
   ομάδαεντολών;
}

voidmain()
{
int a, b, c;
```

- Αν η εντολή κλήσης της συνάρτησης μέσα από το κύριο πρόγραμμα είναι η:

func1(a);

τότε αυτό είναι συντακτικό λάθος γιατί η κλήση της συνάρτησης γίνεται με λιγότερες μεταβλητές από όσες είναι γραμμένες στον ορισμό της συνάρτησης.

- Αν η εντολή κλήσης της συνάρτησης μέσα από το κύριο πρόγραμμα είναι η:

func1(a, b, c);

τότε αυτό είναι συντακτικό λάθος γιατί η κλήση της συνάρτησης γίνεται με περισσότερες μεταβλητές από όσες είναι γραμμένες στον ορισμό της συνάρτησης.

- Αν η εντολή κλήσης της συνάρτησης μέσα από το κύριο πρόγραμμα είναι η:

func1();

τότε αυτό είναι συντακτικό λάθος γιατί η κλήση της συνάρτησης γίνεται χωρίς καμία παράμετρο ενώ στον ορισμό της συνάρτησης περιλαμβάνονται παράμετροι.

Β! Περιορισμός

Ο τύπος των παραμέτρων που γράφεται στην κλήση της συνάρτησης πρέπει να είναι 1 προς 1 ίδιος ή συμβατός με τον τύπο των παραμέτρων που γράφεται στον ορισμό της συνάρτησης.

Παράδειγμα 1

```
void func1(int x, float y, char z)
{
   ομάδα εντολών;
}

voidmain()
{
int a, b, c;
float e, f;
char g, h;
```

- Αν η εντολή κλήσης της συνάρτησης μέσα από το κύριο πρόγραμμα είναι η:

func1(a, b, c);

τότε αυτό είναι συντακτικό λάθος γιατί η κλήση της συνάρτησης γίνεται με 3 ακέραιες παραμέτρους ενώ στον ορισμό της συνάρτησης οι παράμετροι έχουν άλλους τύπους. Συγκεκριμένα όταν καλείται η συνάρτηση func1 γίνονται τα εξής:

Μεταβλητές κύριου προγράμματος	**Παράμετροι func 1**
a →	x σωστό γιατί και οι 2 μεταβλητές είναι του ίδιου τύπου
b →	y σωστό γιατί οι μεταβλητές είναι συμβατού τύπου
c →	z λάθος γιατί οι μεταβλητές είναι διαφορετικού τύπου

Παράδειγμα 2

```
void func1(int x, float y, char z)
{
     ομάδα εντολών;
}

voidmain()
{
```

int a, b, c;
float e, f;
char g, h;

- Αν η εντολή κλήσης της συνάρτησης μέσα από το κύριο πρόγραμμα είναι η:

func1(e, f, g);

τότε αυτό είναι συντακτικό λάθος γιατί η κλήση της συνάρτησης γίνεται με παραμέτρους διαφορετικού τύπου από τις παραμέτρους στον ορισμό της συνάρτησης. Συγκεκριμένα όταν καλείται η συνάρτηση func1 γίνονται τα εξής:

Μεταβλητές κύριου προγράμματος	**Παράμετροι func 1**
e **à**	x σωστό γιατί και οι 2 μεταβλητές είναι συμβατού τύπου
f **à**	y σωστό γιατί οι μεταβλητές είναι συμβατού τύπου
g **à**	z λάθος γιατί οι μεταβλητές είναι διαφορετικού τύπου

14.8 .Παράμετροι δείκτες τύπου *void*

Σε μερικές περιπτώσεις θα θέλαμε να δημιουργήσουμε μια συνάρτηση που θα είναι δυνατόν να δεχθεί παραμέτρους διαφορετικού τύπου σε διαφορετικές κλήσεις. Για παράδειγμα ας υποθέσουμε ότι θέλουμε να ορίσουμε μια συνάρτηση που θα έχει την δυνατότητα να ανταλλάσσει τις τιμές δύο ακεραίων αλλά η ίδια η συνάρτηση να μπορεί να ανταλλάσσει και τις τιμές δύο αριθμών τύπου ***double***. Το πρόβλημα μπορεί να αντιμετωπιστεί κατά ένα τρόπο ορίζοντας τις παραμέτρους της συνάρτησης σαν δείκτες γενικού τύπου. Μια παράμετρος δείκτη γενικού τύπου ορίζεται με τη δήλωση:

void * όνομα_παραμέτρου

Η προσπέλαση κάποιας τιμής χρησιμοποιώντας δείκτη γενικού τύπου γίνεται εφόσον κάθε φορά προσδιορίζουμε στον δείκτη τον συγκεκριμένο τύπο της τιμής που θα προσπελάσει. Αυτό επιτυγχάνεται με τον τελεστή μετατροπής τύπου (cast). Ο τελεστής εκφράζεται με μια παρένθεση πριν από το όνομα του δείκτη μέσα στην οποία αναγράφουμε τον τύπο του δείκτη.

<u>Παράδειγμα</u>

Αν έχουμε ορίσει τυπική παράμετρο ένα pointer γενικού τύπου με την δήλωση: ***void****p, τότε προκειμένου να τοποθετήσουμε τον ακέραιο 10 στη διεύθυνση που δείχνει ο pointer θα πρέπει να δώσουμε την εντολή:

* (***int*** *)p=10;

Ανάλογα θα τοποθετήσουμε την τιμή τύπου ***double*** 10.5 με την εντολή:

* (***double****) p=10.5;

Για να χρησιμοποιήσουμε ένα δείκτη γενικού τύπου σε παράμετρο συνάρτησης πρέπει στο σώμα ορισμού της συνάρτησης να έχουμε και την πληροφορία του τύπου της πραγματικής διεύθυνσης (κατά την κλήση), ώστε να μετατρέψουμε κατάλληλα τον τύπο του δείκτη (μέσα στη συνάρτηση). Η πληροφορία αυτή περνά στην συνάρτηση με μια επιπλέον παράμετρο.

15 Δομές

Η δομή (structure) είναι ένα σύνθετος τύπος δεδομένων στη C ο οποίος ορίζεται σαν ένα σύνολο στοιχείων διαφορετικού τύπου. Τα στοιχεία μιας δομής ονομάζονται πεδία (fields). Για να ορίσουμε μια δομή χρησιμοποιούμε την ακόλουθη δήλωση:

struct όνομα_δομής
{
τύπος1 πεδίο1;

τύπος2 πεδίο2;
……
τύποςν πεδίον;
};

Στη συνέχεια ορίζουμε μεταβλητές του τύπου της δομής με τη δήλωση:

όνομα_δομής όνομα_μεταβλητής;

Για να προσπελάσουμε τα πεδία μιας δομής πρέπει να χρησιμοποιήσουμε τον ακόλουθο συμβολισμό:

όνομα_μεταβλητής.πεδίο

Η μεταβλητή όνομα_μεταβλητής.πεδίο είναι τύπου ίδιο με αυτό του πεδίου.

Παράδειγμα1

```
struct student
{
char name[10];
float grade;
int age;
};

student s;
```

Για να εισάγουμε τιμή στα πεδία της δομής γράφουμε τις εντολές:

```
printf("Give students name: ");
scanf("%s", &s.name);
printf("Give students grade: ");
scanf("%f", &s.grade);
printf("Give students age: ");
scanf("%d", &s.age);
```

Η μεταβλητή s.name είναι τύπου συμβολοσειράς, η μεταβλητή s.grade είναι τύπου ***float*** και η μεταβλητή s.age είναι τύπου integer.

Για να εμφανίσουμε τα πεδία της δομής γράφουμε τις εντολές:

```
printf("%s", s.name);
printf("%f", s.grade);
printf("%d", &s.age);
```

Παράδειγμα2

```
struct student
{
char name[10];
float grade;
int age;
};

student s[10];
voidmain()
{
int i;
```

```
for (i=0;i<10;i++)
{
    printf("Give students name: ");
    scanf("%s", &s[i].name);
    printf("Give students grade: ");
    scanf("%f", &s[i].grade);
    printf("Give students age: ");
    scanf("%d", &s[i].age);
}

for (i=0;i<10;i++)
{
    printf("%s", s[i].name);
    printf("%f", s[i].grade);
    printf("%d", &s[i].age);
}
```

15.1 Δομές και δείκτες

Για να προσπελάσουμε τα στοιχεία μιας δομής χρησιμοποιώντας δείκτες πρέπει να έχει προηγηθεί η δήλωση:

όνομα_δομής *όνομα_μεταβλητής_δείκτη;

και στη συνέχεια να γράψουμε την εντολή:

όνομα_μεταβλητής_δείκτη **à** πεδίο

για να προσπελάσουμε το κάθε πεδίο της δομής. Οποιαδήποτε όμως χρήση κάνουμε σε δείκτες προς δομές πρέπει να προηγηθεί η εντολή:

όνομα_μεταβλητής_δείκτη=new δομή;

προκειμένου να δεσμεύσουμε χώρο στη μνήμη για τη νέα δομή που δημιουργούμε.

Παράδειγμα1

Στο παράδειγμα αυτό το s είναι δείκτης σε δομή.

```
#include<stdio.h>
struct student
{
    char name[10];
    float grade;
    int age;
};

structstudent *s;

voidmain()
{
    int i;
    for (i=0;i<3;i++)
    {
        s=(struct student *)malloc(n*sizeof(student));
        printf("\n Give %d name: ",i+1);
        scanf("%s", &s->name);
        printf("\n Give %d grade: ",i+1);
        scanf("%f", &s->grade);
```

```
		printf("\n Give %d age: ",i+1);
		scanf("%d", &s->age);
	}

	for (i=0;i<3;i++)
	{
		printf("%s", s->name);
		printf("%5.2f",s->grade);
		printf("%5d",s->age);
		printf("\n");
	}
}
```

Παράδειγμα2

Στο παράδειγμα αυτό το s είναι πίνακας με δείκτες σε δομή.

```
#include<stdio.h>
struct student
{
	char name[10];
	float grade;
	int age;
};

student *s[3];

voidmain()
{
	int i;
	for (i=0;i<3;i++)
	{
		s[i]=new student;
		printf("\n Give %d name: ",i+1);
		scanf("%s", &s[i]->name);
		printf("\n Give %d grade: ",i+1);
		scanf("%f", &s[i]->grade);
		printf("\n Give %d age: ",i+1);
		scanf("%d", &s[i]->age);
	}

	for (i=0;i<3;i++)
	{
		printf("%s", s[i]->name);
		printf("%5.2f",s[i]->grade);
		printf("%5d",s[i]->age);
		printf("\n");
	}
}
```

ΒΜΕΡΟΣ – ΓΛΩΣΣΑC++

1 Περιγραφή C++

Η C++είναι μια γενικού σκοπού γλώσσα προγραμματισμού Η/Υ. Είναι μέσου επιπέδου γλώσσα, καθώς περιλαμβάνει έναν συνδυασμό χαρακτηριστικών από γλώσσες υψηλού και χαμηλού επιπέδου.Υποστηρίζει δομημένο, αντικειμενοστραφή και γενικό προγραμματισμό.Η γλώσσα αναπτύχθηκε από τον BjarneStroustruptο 1979 στα εργαστήρια Bellτης AT&T, ως βελτίωση της γλώσσας προγραμματισμού Cκαι αρχικά ονομάστηκε "C with Classes", δηλαδή C με Κλάσεις. Μετονομάστηκε σε C++ το 1983. Οι βελτιώσεις ξεκίνησαν με την προσθήκη κλάσεων, και ακολούθησαν, μεταξύ άλλων, εικονικές συναρτήσεις, υπερφόρτωση τελεστών, πολλαπλή κληρονομικότητα, πρότυπα κ.α.

2 Διαφορές ανάμεσα στη C++ και C

· Η βασική και προφανής διαφορά μεταξύ των δύο γλωσσών που είναι τα αντικειμενοστραφή στοιχεία της C++Συγκεκριμένα η C++ επεκτείνει την έννοια των δομών ορίζοντας κλάσεις και στιγμιότυπα με συγκεκριμένα χαρακτηριστικά

· Στη C++ υπάρχει και ο λογικός(bool) τύπος δεδομένων ως βασικός τύπος

· Η εισαγωγή και εξαγωγή δεδομένων στη C++ δεν γίνεται με τις συναρτήσεις scanf() και printf() αλλά με τα αντικείμενα cin και cout. Βέβαια μπορούν να χρησιμοποιηθούν και οι συναρτήσεις τις C γιατί η C++ αποτελεί επέκταση της C και χρησιμοποιεί όλες τις λειτουργίες της C

· Η δέσμευση μνήμης στη C++ δεν γίνεται με τις malloc(),calloc(),realloc() αλλά με το αντικείμενο new και η διαγραφή δεν γίνεται με την free() αλλά με το αντικείμενο delete

· Στις δομές, στη C επιτρέπεται η δήλωση μόνο μεταβλητών Αντίθετα στη C++, επιτρέπεται και η δήλωση συναρτήσεων (βέβαια με την ύπαρξη κλάσεων οι δομές γίνονται άχρηστες).

· Με χρήση του τελεστή επίλυσης εμβέλειας :: έχουμε την δυνατότητα πρόσβασης και σε μεταβλητές πέρα από τον χώρο εμβέλειας μας ή την πολλαπλή χρήση ονόματος

· Στις παραμέτρους συναρτήσεων μπορούμε να έχουμε default τιμές κατά τον ορισμό της συνάρτησης, τις οποίες μπορούμε να αλλάξουμε μέσω των ορισμάτων ή ακόμη και να παραβλέψουμε.

· Η μεταβίβαση δεδομένων σε συναρτήσεις της C++ μπορεί να γίνει είτε με πέρασμα τιμών είτε με αναφορά με δείκτες και αναφορικές μεταβλητές ενώ στη C γίνεται είτε με πέρασμα τιμών είτε με αναφορά με δείκτες

3 Εντολές Εισόδου – Εξόδου

3.1 Εντολή Εισόδου

Με την εντολή εισόδου μπορούμε να εισάγουμε τιμές σε μεταβλητές. Έχει την ακόλουθη μορφή:

cin>>μεταβλητή; για να εισάγουμε μια τιμή σε μια μεταβλητή

ή

cin>>μεταβλητή1>>μεταβλητή2>>…; για να εισάγουμε πολλές τιμές σε διαφορετικές μεταβλητές

3.2 Εντολή Εξόδου

Με την εντολή εξόδου μπορούμε να εμφανίσουμε στην οθόνη τιμές μεταβλητών και σχόλια. Έχει την ακόλουθη μορφή:

cout<<"σχόλιο"; για εκτυπώσουμε ένα σχόλιο

ή

cout<<μεταβλητή; για να εκτυπώσουμε ένα αποτέλεσμα

ή

cout<<"σχόλιο"<<μεταβλητή; για εκτυπώσουμε ένα σχόλιο και ένα αποτέλεσμα μαζί

3.3 Εντολή #*include*

Γράφοντας την εντολή:

- #***include***<filename> ψάχνει για το αρχείο που καθορίζουμε μόνο στον ειδικό κατάλογο για ***include*** files
- #***include*** "filename" ψάχνει για το αρχείο που καθορίζουμε και στον τρέχοντα κατάλογο αλλά και στο ***include***directory

<u>Παράδειγμα 1-Πρόγραμμα εκτύπωσης σχολίων</u>

```
#include<iostream.h>
voidmain()
{
   cout<<"First program in C++ \n";
   cout<<'\n';
   cout<<"\n Computer\n";
}
```

<u>Παράδειγμα 2 -Πρόγραμμα εκτύπωσης σχολίων</u>

```
#include<iostream.h>
void main()
{
```

```
    cout<<"First program in C++"<<endl;
    cout<<"Computer "<<endl;
}
```

Παράδειγμα 3 -Πρόγραμμα δήλωσης, εκτύπωσης και αρχικοποίησης μεταβλητών

```
#include<iostream.h>
void main()
{
    intx,y;

    x=1; //αρχικοποίηση μεταβλητής x
    y=2; //αρχικοποίηση μεταβλητής y

    //Εκτύπωση αποτελεσμάτων
    cout<< " x = " << x <<endl<<" y = " <<y<<endl;
}
```

4 Συναρτήσεις

4.1 Εξορισμού (Default) Παράμετροι συνάρτησης (Καθιερωμένες παράμαετροι συνάρτησης)

Η C++ μας δίνει τη δυνατότητα (που δεν υπάρχει στη C) να ορίζουμε αρχικές τιμές σε παραμέτρους με δήλωση της μορφής:

τύπος όνομα_τυπικής_παραμέτρου=τιμή

Έχοντας ορίσει σε μια συνάρτηση παραμέτρους με αρχικές τιμές λέμε ότι η συνάρτηση έχει καθιερωμένες (default) παραμέτρους. Οι καθιερωμένες παράμετροι τοποθετούνται σαν τελευταίες παράμετροι της συνάρτησης. Η συνάρτηση μπορεί να κληθεί με όλες ή με παράλειψη μερικών από τις τελευταίες καθιερωμένες παραμέτρους της. Οι παραλειπόμενες παράμετροι κατά την κλήση θα έχουν ως τιμή την αρχική τιμή που προσδιορίστηκε (την καθιερωμένη). Με τον τρόπο αυτό μπορούμε να καλούμε μια συνάρτηση με διαφορετικό πλήθος παραμέτρων από κλήση σε κλήση.

Σημείωση

Η αρχικοποίηση μιας default παραμέτρου μπορεί να γίνει με οποιαδήποτε τιμή και όχι υποχρεωτικά με μηδέν

Παράδειγμα 1

Στο παράδειγμα αυτό ορίζουμε μια συνάρτηση με το όνομα sum με default παραμέτρους

```
#include<iostream.h>

double sum(int k=0, int m=0) //Η συνάρτηση sum μπορεί να κληθεί με 0, 1 ή 2 παραμέτρους
{
    cout<<"A: ";
    return k+m;
}

double sum(int k, int m, int n) //Η συνάρτηση sum μπορεί να κληθεί μόνο με 3 παραμέτρους
{
    cout<<"B: ";
    return k+m+n;
}

voidmain()
{
    cout<<"sum() = "<<sum()<<endl; //καλείταιηπρώτησυνάρτηση sum()
    cout<<"sum(10) = "<<sum(10)<<endl; //καλείται η πρώτη συνάρτηση sum()
    cout<<"sum(10,20) = "<<sum(10,20)<<endl; //καλείται η πρώτη συνάρτηση sum()
    cout<<"sum(10,20,30) = "<<sum(10,20,30)<<endl; //καλείται η δεύτερη συνάρτηση sum()
}
```

Αποτελέσματα Προγράμματος

```
A: sum()=0
A: sum(10)=10
A: sum(10,20)=30
B: sum(10,20,30)=60
```

Βασική Παρατήρηση
Η υπερφόρτωση δεν θα μπορούσε να γίνει με την τρίτη παράμετρο να έχει default τιμή, διότι καλώντας τη συνάρτηση sum με 2 παραμέτρους δεν θα ήταν δυνατό να καθοριστεί ποια από τις 2 συναρτήσεις πρέπει να εκτελεστεί. Συγκεκριμένα αν επιχειρούσαμε να το κάνουμε αυτό θα προέκυπτε συντακτικό σφάλμα.

Παράδειγμα 2
Στο παράδειγμα αυτό ορίζουμε τη συνάρτηση με το όνομα showchar με default παραμέτρους οι οποίες διαφέρουν μεταξύ τους μόνο ως προς τη σειρά που γράφονται. Ακόμα και έτσι το κύριο πρόγραμμα γνωρίζει ποια συνάρτηση να καλέσει

```
#include<iostream.h>

void showchar(int n, char a) //Η συνάρτηση showchar μπορεί να κληθεί κατά σειρά πρώτα με ακέραια παράμετρο και μετά με παράμετρο χαρακτήρα
{
    for (int i=0; i<n; i++)
        cout<<a;
}

void showchar(char a, int n) //Η συνάρτηση showchar μπορεί να κληθεί κατά σειρά πρώτα με παράμετρο χαρακτήρα και μετά με ακέραια παράμετρο
{
    for (int i=0; i<n; i++)
        cout<<a;
}

voidmain()
{
    showchar(3,'A'); //κλήση με ακέραιο, χαρακτήρα (καλείται η πρώτη συνάρτηση showchar)
    cout<<endl;
    showchar('A',3); //κλήση με χαρακτήρα, ακέραιο (καλείται η δεύτερη συνάρτηση showchar)
}
```

Βασική Παρατήρηση
Όταν καλούμε συνάρτηση και οι πραγματικές παράμετροι δεν έχουν τον τύπο των τυπικών παραμέτρων τότε γίνεται μετατροπή (αν είναι επιτρεπτή) της πραγματικής τιμής σε τιμή που έχει τον τύπο της τυπικής παραμέτρου. Στη C++ κάθε ακέραιος επιτρέπεται να μετατραπεί σε τύπου χαρακτήρα και το αντίστροφο. Η περίπτωση που θα χρειαστούν μετατροπές στις πραγματικές παραμέτρους μπορεί να προκαλέσει σφάλμα. Για παράδειγμα ας υποθέσουμε ότι καλούμε τη συνάρτηση showchar με την εντολή showchar(10,10); Τότε δίνουμε και τις 2 παραμέτρους ακέραιου τύπου. Δεν έχουμε όμως ορίσει τέτοια υπερφόρτωση. Θα επιχειρηθεί να γίνει μετατροπή των παραμέτρων ώστε να ταιριάζουν με τον τύπο των τυπικών παραμέτρων των συναρτήσεων που έχουν οριστεί. Τέτοιο ταίριασμα όμως μπορεί να γίνει και στις 2 συναρτήσεις (αφού είπαμε ότι ο τύπος char μπορεί να μετατραπεί σε int και το αντίστροφο). Στην περίπτωση αυτή δεν είναι ξεκάθαρο ποια από τις 2 συναρτήσεις πρέπει να κληθεί και ο compiler θα δώσει σφάλμα αμφιβολίας (ambiguity error).

Παράδειγμα 3

```
#include<iostream.h>

double sum (double a=0, double b=0, double c=0, double d=0) //όλεςοιπαράμετροιέχουν default τιμέςμηδέν
{
   return a+b+c+d;
}

void main()
{
   cout<<"sum() = "<<sum()<<endl;
   cout<<"sum(100.5) = "<<sum(100.5)<<endl;
   cout<<"sum(10,20) = "<<sum(10,20)<<endl;
   cout<<"sum(10,20,-5.5) = "<<sum(10,20,-5.5)<<endl;
   cout<<"sum(1,2,3,4) = "<<sum(1,2,3,4)<<endl;
}
```

4.2 Υπερφόρτωση Συναρτήσεων

Ένα από τα πλέον δυναμικά χαρακτηριστικά της C++ είναι η δυνατότητα να χρησιμοποιούνται διαφορετικές συναρτήσεις με το ίδιο όνομα. Στη C++ μια συνάρτηση αναγνωρίζεται όχι μόνο από το όνομα της αλλά και από το πλήθος, τον τύπο και τη σειρά των παραμέτρων της. Δύο ή περισσότερες συναρτήσεις μπορεί να μοιράζονται το ίδιο όνομα σε ένα πρόγραμμα. Ορίζοντας μια νέα συνάρτηση με όνομα παλιάς λέμε ότι δημιουργούμε **υπερφόρτωση (overloading) της παλιάς συνάρτησης**. Αυτό γίνεται όταν θέλουμε η ίδια συνάρτηση να εκτελεί **διαφορετικές λειτουργίες ανάλογα με τα ορίσματα που λαμβάνει**. Αν σε ένα πρόγραμμα έχουμε υπερφορτώσεις μιας συνάρτησης και θέλουμε να καλέσουμε μια από αυτές τότε θα πρέπει να δώσουμε μετά το όνομα της πραγματικές παραμέτρους που αντιστοιχίζονται (ως προς το πλήθος, τύπο και τη σειρά) με τις τυπικές παραμέτρους.

Παράδειγμα 1-Πρόγραμμα με υπερφόρτωση συναρτήσεων

```
#include<iostream.h>


//Ορισμός 1ης Υπερφόρτωσης της συνάρτησης sum

double sum(int k=0,int m=0)

{

//Αφού η συνάρτηση sum έχει καθιερωμένες αρχικές τιμές μπορεί να κληθεί είτε με μια είτε με δύο είτε με καμία παράμετρο

        cout<<"A: ";

        return k+m;

}
```

```
//Ορισμός 2ης Υπερφόρτωσης της συνάρτησης sum
double sum(int k,int m,int n)
{
        cout<<"B: ";
        return k+m+n;
}

void main()
{
      cout<<"sum()="<<sum()<<'\n';//Καλείται η πρώτη sum
      cout<<"sum(10)="<<sum(10)<<'\n'; //Καλείται η πρώτη sum
      cout<<"sum(10,20)="<<sum(10,20)<<'\n'; //Καλείται η πρώτη sum
      cout<<"sum(10,20,30)="<<sum(10,20,30)<<'\n'; //Καλείται η δεύτερη sum
      //Στο πρόγραμμα αυτό έχουμε υπερφόρτωση συναρτήσεων γιατί διαφορετικές συναρτήσεις μοιράζονται το ίδιο όνομα
}
//Αποτελέσματα προγράμματος:
//A: sum()=0
//A: sum(10)=10
//A: sum(10,20)=30
//B: sum(10,20,30)=60
```

5 Αναφορές

Η αναφορά είναι ένα ψευδώνυμο μιας μεταβλητής. Συμβολίζεται με το χαρακτήρα &. Πρέπει κατά τη δήλωση της αναφοράς να γίνεται ταυτόχρονα και αρχικοποίηση προς τη μεταβλητή της οποίας αποτελεί ψευδώνυμο.Η δήλωση μιας αναφοράς έχει την ακόλουθη γενική μορφή:

τύπος &όνομα αναφοράς = μεταβλητή;

π.χ.

```
int&rx= x;
```

Παράδειγμα 1-Πρόγραμμα με αναφορά προς μεταβλητή

```
#include<iostream.h>
```

```
void main()
{
    int a=3,&p=a,b=a;//δηλώνονται μεταβλητή a, εναλλακτικό όνομα p της a και μεταβλητή b με αρχική τιμή την a

    p++;//τροποποίηση της p (αλλάζει την τιμή της a)
    cout<<a<<" "<<p<<endl; //τυπώνονται 4 (από την a) και 4 (από την p)
    b=b+10; // τροποποίηση της b (δεν αλλάζει την τιμή της a)
    cout<<a<<" "<<b<<endl; //τυπώνονται 4 (από την a) και 13 (από την b)
}
//Αποτελέσματα προγράμματος:
//4 4
//4 13
```

Παράδειγμα 2-Πρόγραμμα με αναφορά προς δείκτη

```
#include<iostream.h>

void main()
{
    int a=10,b=50;//ορίζονται οι μεταβλητές a και b
    int *q=&a;//ορίζεται ο pointerq προς τη μεταβλητή a
    int *&r=q;//ορίζεται αναφορά (εναλλακτικό όνομα) προς τον pointerq

    cout<<*r<<endl; //έμμεση τιμή με το εναλλακτικό όνομα είναι η τιμή της a
    r=&b//αλλάζουμε τη διεύθυνση που δείχνει ο r
    cout<<*q<<endl; //εκτυπώνεται σαν έμμεση τιμή του q η τιμή της b
}
//Αποτελέσματα προγράμματος:
//10 50
```

Παράδειγμα 3-Πρόγραμμα με πέρασμα τιμής, αναφοράς και δείκτη

```
#include<iostream.h>

void swap(int x,int y); //κλήση της swap με τιμή
void swap1(int *px,int *py); //κλήση της swap1 με δείκτες
void swap2(int&rx,int&ry); //κλήση της swap2 με αναφορές

void main()
{
    int x=5,y=10;

    cout<<"Before swap x = "<<x<<" and y ="<<y<<"\n";
    swap(x,y); //pass by value. Δημιουργούνται τοπικά αντίγραφα των x και y στη συνάρτηση
    cout<<"After swap x = "<<x<<" and y ="<<y<<"\n";

    x=5;
    y=10;
    cout<<"\n\nBefore swap1 x = "<<x<<" and y ="<<y<<"\n";
    swap1(&x,&y); //pass by reference with pointers. Οι διευθύνσεις των x και y περνάνε στο υποπρόγραμμα
    cout<<"After swap1 x = "<<x<<" and y ="<<y<<"\n";

    x=5;
    y=10;
    cout<<"\n\nBefore swap2 x = "<<x<<" and y ="<<y<<"\n";
    swap2(x,y); //pass by reference with references. Οι τιμές των x και y δεν καταχωρούνται σε τοπικές μεταβλητές αλλά σε ψευδώνυμα
    cout<<"After swap2 x = "<<x<<" and y ="<<y<<"\n";
}

void swap(int x,int y)
{
    int temp;
```

```
        temp=x;
        x=y;
        y=temp;
        cout<<"In swap x = "<<x<<" and y = "<<y<<"\n";
        }
        void swap1(int *px,int *py)
        {
        int temp;

        temp=*px;
        *px=*py;
        *py=temp;
        cout<<"In swap1 *px = "<<*px<<" and *py = "<<*py<<"\n";
}

void swap2(int&rx,int&ry)
{
        int temp;

        temp=rx;
        rx=ry;
        ry=temp;
        cout<<"In swap2 rx = "<<rx<<" and ry = "<<ry<<"\n\n";
}
//Αποτελέσματα προγράμματος
//Beforeswapx=5 andy=10
//In swap x=10 and y=5
//After swap x=5 and y=10

//Before swap1 x=5 and y=10
//In swap1 *px=10 and *py=5
```

//**After swap1 x=10 and y=5**

//**Before swap2 x=5 and y=10**

//**In swap2 rx=10 and ry=5**

//**After swap2 x=10 and y=5**

6 Δυναμική Διαχείριση Μνήμης

Η C++ επιτρέπει τη δυναμική διαχείριση της μνήμης. Αυτό σημαίνει ότι μπορούμε να δεσμεύουμε όση μνήμη (κύρια μνήμη) χρειαζόμαστε δυναμικά δηλαδή κατά τη διάρκεια εκτέλεσης του προγράμματος και όχι στατικά δηλαδή εκ των προτέρων. Η δυναμική δέσμευση της μνήμης γίνεται με την εντολή ***new***. Ως αποτέλεσμα η εντολή ***new*** επιστρέφει τη διεύθυνση του 1ου byte από τη μνήμη που δεσμεύεται. Για να αποδεσμεύσουμε τη μνήμη που δεν μας χρειάζεται χρησιμοποιούμε την εντολή ***delete***. Η δυναμική δέσμευση μνήμης είναι ιδιαίτερα χρήσιμη στους πίνακες γιατί δεσμεύοντας ακριβώς τη μνήμη που μας χρειάζεται για ένα πίνακα αποφεύγουμε 2 προβλήματα:

- Η δεσμευόμενη μνήμη να μην επαρκεί γα τον πίνακα
- Η δεσμευόμενη μνήμη να είναι υπερβολικά μεγάλη σε σχέση με αυτή που πραγματικά χρειάζεται

Παράδειγμα 1 -Πρόγραμμα με δυναμική κατανομή μνήμης για μονοδιάστατο πίνακα

```
#include<iostream.h>
#include<iomanip.h>//Βρίσκεται η setw()
#include<stdlib.h>//Βρίσκεται η exit()
voidmain()
{
int i,m,*a;//H m δείχνει τη διάσταση του πίνακα

//ΒΑΣΙΚΟ!! Στους στατικούς πίνακες η διάσταση με την οποία δηλώνουμε τον πίνακα πρέπει να είναι σταθερά. Αντίθετα κατά τον ορισμό του δυναμικού πίνακα η διάσταση μπορεί να είναι μια μη σταθερή μεταβλητή, αλλά να διαβάζεται κατά την εκτέλεση του προγράμματος όπως γίνεται για το m

        cout<<"Δώσε το μέγεθος του πίνακα: ";
        cin>>m;//Το μέγεθος του πίνακα διαβάζεται κατά τη διάρκεια εκτέλεσης του προγράμματος δεν δίνεται εκ των προτέρων

        a=newint[m];//Δεσμεύεται χώρος για τον πίνακα στο heap
        //Βασικό! Η ισοδύναμη εντολή στη C είναι a=(int)malloc(m*sizeof(int));
        if (!a)//Εάν δεν έγινε κατανομή χώρου
        {
                cout<<"ΛΑΘΟΣ ΚΑΤΑΝΟΜΗΣ";
                exit(1);
        }
        for (i=0;i<m;i++)
        {
                cout<<"Δώσε το "<<i+1<<" στοιχείο του πίνακα: ";
                cin>>a[i];
```

```
        }
        for (i=0;i<m;i++)
                cout<<setw(4)<<*(a+i)<<" ";//Εκτυπώνονται τα στοιχεία του πίνακα
                cout<<'\n';
        delete []a;//Απελευθερώνεται η μνήμη του heap
        //Βασικό! Η ισοδύναμη εντολή στη C είναι free(a);
}
```

Παράδειγμα 2 - Πρόγραμμα με δυναμική κατανομή μνήμης για δισδιάστατο πίνακα

```
#include<iostream.h>
#include<iomanip.h>//για την setw()
#include<stdlib.h>//για την exit()
voidmain()
{
int i,j,m,n;
```

//ΠΡΟΣΟΧΗ! Όπως στο μονοδιάστατο έτσι και στο δι-διάστατο πίνακα το πλήθος των γραμμών και των στηλών του δεν ορίζεται εκ των προτέρων (στατικά) αλλά διαβάζονται τη στιγμή εκτέλεσης του προγράμματος (δυναμικά)

```
        cout<<"Δώσε το πλήθος γραμμών του πίνακα: ";
        cin>>m;

        cout<<"Δώσε το πλήθος στηλών του πίνακα:  ";
        cin>>n;
```

//Ο δι-διάστατος πίνακας δεν μπορεί να αναπαρασταθεί απευθείας στο heap οπότε για την αναπαράσταση του γίνονται τα ακόλουθα βήματα:

//Βήμα 1

```
        int *p=newint[m*n];
```

//Ορίζεται ο pointerp προς ακεραίους, κρατούνται θέσεις μνήμης στο heap για m*n ακεραίους και η διεύθυνση του πρώτου ακέραιου τοποθετείται στο p

//Βήμα 2

```
        int **a=newint *[m];
```

//Ορίζεται ο a σαν δείκτης προς δείκτη σε ακέραιο, κρατούνται m θέσεις στο heap για διευθύνσεις γραμμών (όσες είναι και οι γραμμές) και η πρώτη διεύθυνση αυτών των θέσεων τοποθετείται στο a

```
        if (!p ||!a)//Εάν δεν έγινε κάποια κατανομή μνήμης έξοδος
                exit(1);
```

//Βήμα 3

```
        for(i=0;i<m;i++)
                a[i]=p+i*n;
```

//Προσδιορίζουμε ότι η διεύθυνση a[0] (πρώτο στοιχείο του a) θα έχει τιμή p (διεύθυνση πρώτης γραμμής του πίνακα a), η διεύθυνση a[1] (δεύτερο στοιχείο του πίνακα a) θα είναι η διεύθυνση μετά από n θέσεις από την p στον πίνακα p (δηλαδή η διεύθυνση στον p όπου βρίσκεται το πρώτο στοιχείο της επόμενης γραμμής) κ.λ.π.

```
cout<<"Διάβασμα του πίνακα"<<endl;
for (i=0;i<m;i++)
        for (j=0;j<n;j++)
        {
                cout<<"Διάβασε το "<<i+1<<" "<<j+1<<" στοιχείο του πίνακα: ";
                cin>>a[i][j];
        }

cout<<"Εκτύπωση του πίνακα"<<endl;
for (i=0;i<m;i++)
{
        for (j=0;j<n;j++)
                cout<<setw(4)<<a[i][j]<<" ";
        cout<<'\n';
}

        delete []p;//Απελευθερώνεται η μνήμη όπου είναι οι ακέραιοι
        delete []a;//Απελευθερώνεται η μνήμη όπου είναι οι διευθύνσεις των πρώτων στοιχείων των γραμμών
}
```

7 Σύνθετοι Τύπου Δεδομένων

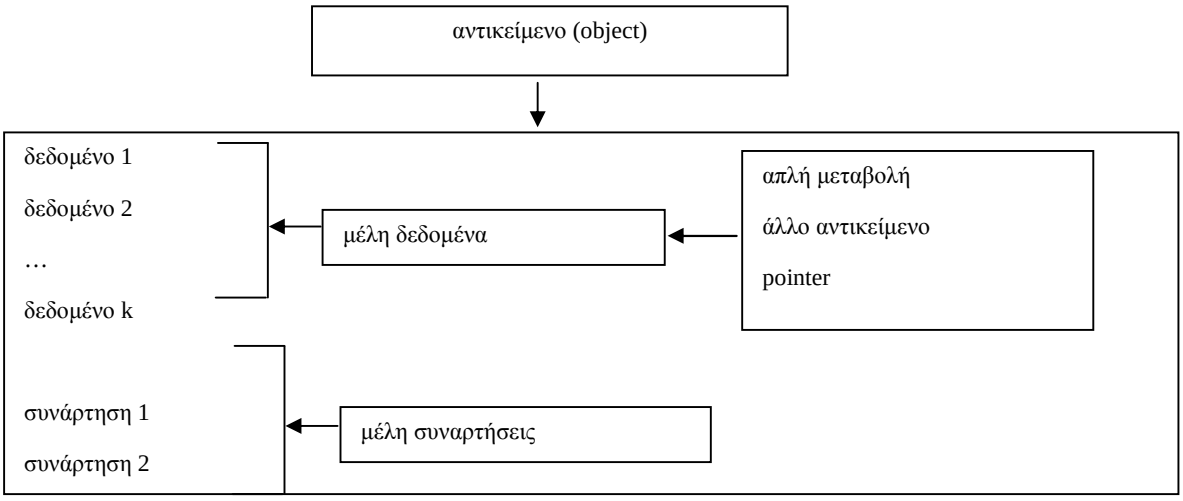
7.1 Βασικές Έννοιες

7.1.1 Τα Αντικείμενα (objects) και τα μέλη τους

Ένα από τα βασικά χαρακτηριστικά της C++ είναι η δυνατότητα δημιουργίας **αντικειμένων** (objects). Τα αντικείμενα είναι χώρος μνήμης όπου κρατούνται ενότητες από δεδομένα διαφόρων τύπων και συναρτήσεις σχετικές με τα δεδομένα.

αντικείμενα = χώρος μνήμης με δεδομένα + συναρτήσεις για τα δεδομένα

Τα δεδομένα ενός αντικειμένου μπορεί να είναι μεταβλητές απλών τύπων, πίνακες (arrays), άλλα αντικείμενα ή μεταβλητές διευθύνσεων (pointers) προς μεταβλητές, συναρτήσεις ή άλλα αντικείμενα. **Οι συναρτήσεις και τα δεδομένα ενός αντικειμένου ονομάζονται μέλη** (members) του αντικειμένου. Θέτοντας τα δεδομένα και τις συναρτήσεις που αφορούν αυτά σε αντικείμενα (objects) επιτυγχάνουμε να προφυλάσσουμε τα δεδομένα από ανεπιθύμητες τροποποιήσεις και να αναπτύσσουμε μεγάλα προβλήματα με ευκολία.



Τα αντικείμενα σε ένα πρόγραμμα μπορεί να είναι **επώνυμα** (να έχουν όνομα) ή ανώνυμα (να προσδιορίζονται από δείκτη.

7.1.2 Κατηγορίες και Τύποι Αντικειμένων

Η C++ προσφέρει τη δυνατότητα δημιουργίας τριών κατηγοριών αντικειμένων. Αντικείμενα κατηγορίας **ένωσης** (union), κατηγορίας **δομής** (structure) και κατηγορίας **κλάσης** (***class***). Για κάθε μια κατηγορία μπορούμε να δημιουργήσουμε αντικείμενα διαφορετικού τύπου. Ο τύπος προσδιορίζει το "καλούπι" πάνω στο οποίο δημιουργούνται τα αντικείμενα και συνήθως χαρακτηρίζεται στο πρόγραμμα από ένα όνομα. Για κάθε τύπο (μιας κατηγορίας) μπορούμε να δημιουργούμε ένα ή περισσότερα αντικείμενα.

7.1.3 Βήματα για Δημιουργία Αντικειμένων

Για να χρησιμοποιήσουμε αντικείμενα στο πρόγραμμά μας ακολουθούμε 2 φάσεις:

- Ορίζουμε ένα σύνθετο τύπο ο οποίος θα ανήκει σε μια από τις κατηγορίες ένωσης (union), δομή (structure), κλάση (***class***).
- Ορίζουμε αντικείμενα (objects) με βάση τον τύπο.

Για παράδειγμα μπορεί να χρησιμοποιούμε στο πρόγραμμά μας δύο αντικείμενα διαφορετικού τύπου της κατηγορίας κλάσης στα οποία το ένα να έχει μέλη ένα ακέραιο και δύο συναρτήσεις. και το άλλο να έχει δύο μέλη ακεραίους και μια συνάρτηση. Τα αντικείμενα έχουν οπωσδήποτε κάποιο τύπο (επώνυμο ή ανώνυμο).

7.1.4 Βασικές Διαφορές κατηγοριών Αντικειμένων

Η διαφορά των κατηγοριών κλάσεων και δομών στη C++ δεν είναι ουσιαστική. Αφορά μόνο τον καθιερωμένο τρόπο προσπέλασης των μελών των αντικειμένων.

7.1.5 Προσπέλαση και Άδειες Προσπέλασης Μελών

Σε ένα πρόγραμμα που έχουμε ορίσει ένα αντικείμενο με κάποια μέλη μπορούμε να προσπελαύνουμε τα μέλη, μέσω του ονόματός του αντικειμένου ή μέσω ενός pointer στον οποίο εκχωρούμε την διεύθυνση του αντικειμένου.
Ας υποθέσουμε ότι έχουμε ορίσει ένα αντικείμενο με όνομα a και με μέλη μια συνάρτηση με όνομα f και μια μεταβλητή με όνομα k. Ας υποθέσουμε ακόμη ότι, έχουμε ορίσει ένα pointerp (προς αντικείμενα του τύπου του a) και έχουμε εκχωρήσει (με p=&a) στον p την διεύθυνση του αντικειμένου a. Τότε η προσπέλαση των μελών του a γίνεται:

α) ***χρησιμοποιώντας το όνομα και τον τελεστή "." (τελεία) με τις παραστάσεις***:

- a.k (για τον προσδιορισμό του μέλους δεδομένου)
- a.f() (για την κλήση της συνάρτησης μέλους)

β)**χρησιμοποιώντας τον pointer και τον τελεστή "->"με τις παραστάσεις**:

- p->k (για τον προσδιορισμό του μέλους δεδομένο)
- p->f () (για την κλήση της συνάρτησης μέλους)

Στη C++ τα μέλη των αντικειμένων μπορεί να χαρακτηρίζονται (από τον τύπο των αντικειμένων) σαν **δημόσια** (***publc***) ή **ιδιωτικά** (***private***) ή **προστατευμένα** (***protected***).

Σημείωση:
Οι τελεστές τελεία "." και "->" χρησιμοποιούνται μόνο για την προσπέλαση μελών που είναι δημόσια (***publc***). Δεν επιτρέπεται η χρησιμοποίησή τους σε μέλη ιδιωτικά (***private***) ή προστατευμένα (***protected***).

7.1.6 Ορισμός Συνθετών Τύπων Αντικειμένων

Για να ορίσουμε ένα αντικείμενο πρέπει προηγούμενα να ορίσουμε τον τύπο (το καλούπι) του αντικειμένου τον οποίο αποκαλούμε **σύνθετο τύπο δεδομένων**. Στη C++ έχουμε την δυνατότητα να ορίσουμε τρεις κατηγορίες σύνθετων τύπων. Σύνθετους τύπους (καλούπια) για **ενώσεις**, για **δομές** και για **κλάσεις**. Οι σύνθετοι τύποι (για κάθε κατηγορία) ορίζονται με τον ίδιο τρόπο και με δηλώσεις στις οποίες διακρίνουμε τρία τμήματα:**Επικεφαλίδα, Σώμα τύπου, Λίστα αντικειμένων**

- Στην επικεφαλίδα αναφέρεται η κατηγορία του τύπου με μια από τις δεσμευμένες λέξεις union (για ενώσεις), ***struct*** (για δομές), ***class*** (για κλάσεις) και το όνομα του τύπου. (Το όνομα του τύπου μπορεί να μην υπάρχει στην περίπτωση που τα αντικείμενα που θα δημιουργηθούν, είναι μόνο εκείνα που αναφέρονται στην λίστα αντικειμένων).

- Στο σώμα τύπου διακρίνουμε μέσα σε μπλοκ { } δηλώσεις με τις οποίες:
  1. περιγράφονται τα μέλη δεδομένα
  2. δηλώνονται τα πρωτότυπα ή ορίζονται οι συναρτήσεις μέλη.
  3. προσδιορίζονται οι άδειες προσπέλασης των μελών (μια από τις δεσμευμένες λέξεις: ***publc*****:** ή ***private*****:** ή ***protected*****:** πριν από μέλος ή τα μέλη)

- Στη λίστα αντικειμένων αναφέρονται ονόματα αντικειμένων. Τα ονόματα αντικειμένων αν είναι πολλά διαχωρίζονται με κόμμα. Η λίστα επιτρέπεται να είναι κενή. Αν η λίστα είναι κενή, ο σύνθετος τύπος πρέπει να έχει οπωσδήποτε όνομα και τα αντικείμενα να ορίζονται σε άλλα σημεία του προγράμματος, με εντολές δήλωσης τύπου: όνομα_σύνθετου τύπου όνομα_αντικειμένου;

Οι συναρτήσεις μέλη του τύπου μπορεί να ορίζονται μέσα στο μπλοκ σώματος του τύπου ή μέσα στο μπλοκ να αναφέρεται μόνο το πρωτότυπό τους και ο ορισμός να δίνεται εκτός του μπλοκ. Εάν μια συνάρτηση ορίζεται μέσα στο μπλοκ (του σύνθετου τύπου) χαρακτηρίζεται σαν εντός γραμμής (**inline**). Μια συνάρτηση **inline** αναπτύσσει κώδικα (εντολές) σε κάθε σημείο του προγράμματος που θα κληθεί. **Χρησιμοποιούνται inline συναρτήσεις όταν ο κώδικας είναι μικρός για να επιταχύνεται η εκτέλεση του προγράμματος** (διότι δεν γίνεται αναζήτηση του σημείου εισόδου της συνάρτησης και καθυστέρηση για μεταφορά ελέγχου του προγράμματος). Όταν ο κώδικας είναι "μεγάλος" τότε με inline συνάρτηση θα έχουμε "μεγάλη" αύξηση του πλήθος εντολών του προγράμματος άρα και η πιθανή επιβράδυνση της εκτέλεσης. Δεν επιτρέπεται μια inline συνάρτηση να έχει στον ορισμό της ανακυκλώσεις (***for***, ***while***, ***dowhile***) ή διακλαδώσεις (***if***, ***case***). Μια τέτοια συνάρτηση θα ορισθεί εκτός γραμμής (outofline). Στο μπλοκ ορισμού του σύνθετου τύπου θα δοθεί μόνο το πρωτότυπο της συνάρτησης και ο ορισμός θα γίνει εκτός μπλοκ. Μια συνάρτηση μέλος ορίζεται εκτός του μπλοκ περιγραφής των μελών με εντολές της μορφής:
τύπος_συνάρτηση όνομα_σύνθετου_τύπου :: όνομα_συνάρτησης {…εντολές…}

Σημείωση

Μια συνάρτηση μέλος μπορεί να οριστεί σαν inline και εκτός του μπλοκ ορισμού του τύπου, αν βάλουμε πριν τον ορισμό της τη λέξη κλειδί inline με εντολές της μορφής:
inline τύπος_συνάρτησης όνομα_τύπου :: όνομα_συνάρτησης {…εντολές…}

7.1.7 Η Ενθυλάκωση (Encapsulation) Μελών

Δημιουργώντας ένα σύνθετο τύπο έχουμε τη δυνατότητα να προσδιορίζουμε κάποια μέλη σαν **ιδιωτικά** (***private***). Αυτό παρέχει την ευκολία να αποκρύπτουμε τις λεπτομέρειες (βοηθητικές ή μεταβλητές συναρτήσεις) που συμμετέχουν στην εκτέλεσης της διαδικασίας που μας ενδιαφέρει. Έχουμε όπως λέμε τη δυνατότητα πιο καθαρής διασύνδεσης (interface) με το αντικείμενο και επομένως αποφυγή λαθών λόγω πιθανώς μη επιτρεπτών παρεμβάσεων στα μέλη του.

Με το χαρακτηρισμό ενός μέλους **σαν ιδιωτικό επιτυγχάνουμε την απόκρυψη του μέλους (datahiding) από αυτόν που χρησιμοποιεί το αντικείμενο**. **Είναι σαν να τοποθετούμε τα ιδιωτικά μέλη σε μια ειδική θέση και να τα προφυλάσσουμε από εξωτερικές επιδράσεις**. Η ιδιότητα αυτή αναφέρεται σαν **ενθυλάκωση** (encapsulation) των μελών στο αντικείμενο.

7.1.8 Καθιερωμένες Προσπελάσεις

Έχουμε αναφέρει, ότι οι άδειες προσπέλασης που μπορούμε να εκχωρήσουμε σε μέλη αντικειμένων κατά τον προσδιορισμό ενός σύνθετου τύπου, είναι τρεις:

publc (δημόσια), ***private*** (ιδιωτικά) και ***protected*** (προστατευμένα)

@ Όταν το μέλος είναι *publc* (δημόσιο) επιτρέπεται η προσπέλασή του με έκφραση της μορφής a.k

@ Όταν το μέλος είναι *private* (ιδιωτικό) ή *protected* (προστατευμένο) μια τέτοια προσπέλαση δεν είναι επιτρεπτή

Ο χαρακτηρισμός ενός μέλους σαν ***protected*** χρησιμοποιείται σε παραγόμενους σύνθετους τύπους. Η C++ έχει καθιερωμένες (default) άδειες προσπέλασης των μελών, για αντικείμενα των τύπων των κατηγοριών ενώσεων, δομών και κλάσεων σε περίπτωση που δεν χαρακτηρίζουμε άμεσα την άδεια προσπέλασης. Εάν δεν καθορίζουμε άμεσα την άδεια προσπέλασης μέλους (με τις δεσμευμένες λέξεις ***publc*** ή ***protected*** ή ***private***) τότε κατά τα καθιερωμένα:

· τα μέλη αντικειμένων των κατηγοριών ενώσεων (union) και δομών (structure) είναι ***publc*** (δημόσια)

· τα μέλη αντικειμένων της κατηγορίας κλάση (***class***) είναι ***private*** (ιδιωτικά)

Βασική Διαφορά

Η μοναδική διαφορά μεταξύ κλάσης (***class***) και δομής (structure) είναι ότι τα μέλη αντικειμένων της πρώτης κλάσης προκαθορίζονται ***private*** (ιδιωτικά) ενώ της δομής ***publc*** (δημόσια).

7.1.9 Παραδείγματα Ενώσεων, Δομών και Κλάσεων

Σε κάθε στήλη του πίνακα που ακολουθεί ορίζουμε και ένα σύνθετο τύπο δεδομένων. Στην $1^{η}$ στήλη ορίζεται ανώνυμος τύπος κατηγορίας ένωση και αντικείμενα a, b. Στη $2^{η}$ στήλη ορίζεται τύπος s κατηγορίας δομή. Στην $3^{η}$ στήλη τύπος c κατηγορίας κλάση με αντικείμενο το r.

union	***struct*** s	***class*** c
{	{	{
int k;	***int*** k;	***int*** k;

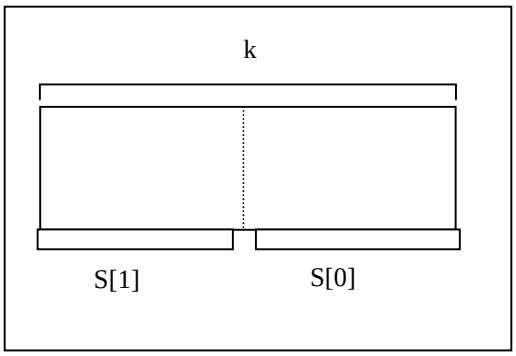
char s [2];	***void*** show () {***cout***<<k;}	***publc***:
} a, b;	};	***void*** put (***int*** x) {x=x;}
		void show ()
		} r;
		void c:: show () {***cout***<<k;}

Ειδικότερα για κάθε στήλη διαπιστώνουμε:

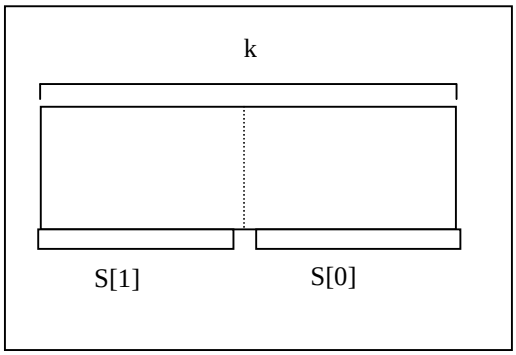
Στην 1η στήλη

Τα αντικείμενα a, b του τύπου έχουν δύο μέλη δεδομένα. Το μέλος k (ακέραιος) και το μέλος s (πίνακας με 2 χαρακτήρες). Επειδή ο τύπος είναι ανώνυμος δεν είναι δυνατόν να οριστούν άλλα αντικείμενα εκτός των a, b. **Τα μέλη των αντικειμένων αυτού του τύπου είναι *publc* (κατά τα καθιερωμένα για τις ενώσεις).** Τα μέλη του a μοιράζονται τον ίδιο χώρο μνήμης και το αυτό συμβαίνει για τα μέλη του αντικειμένου b.

αντικείμενο a



αντικείμενο b



Το αντικείμενο a έχει τα μέλη a.k και τον πίνακα με στοιχεία a.s [0] και a.s [1] που μοιράζονται τον ίδιο χώρο μνήμης. Το b έχει επίσης τα μέλη b.k και πίνακα με στοιχεία b.s[0] και b.s[1]. Μπορούμε μέσα από μια συνάρτηση να τοποθετούμε στο ένα μέλος του αντικειμένου a (ή του b) μια τιμή και να εμφανίζουμε την τιμή του άλλου μέλους π.χ. να δίνουμε τιμή στο k και να εκτυπώνουμε τα bytes από τα στοιχεία του s ή το αντίστροφο να δίνουμε τους χαρακτήρες στο s και να παίρνουμε την τιμή του διαμορφούμενου ακεραίου στο k (δηλαδή να δώσουμε τις εντολές a.k =100; ***cout***<<a.s [0] << " "<<a.s [1] ή τις εντολές b.s[0] ='A' ;b.s [1] ='B'; ***cout***<<b.k;)

Στη 2η στήλη

Με τον ορισμό του τύπου s δεν ορίζονται και αντικείμενα (η λίστα είναι κενή). Αν ορισθούν κάπου αλλού στο πρόγραμμά μας αντικείμενα του τύπου s, θα έχουν μέλη ένα ακέραιο k και μια inline συνάρτηση show() που θα εμφανίζει την τιμή του ακεραίου. **Και τα δύο μέλη είναι *publc* (δημόσια) κατά τα καθιερωμένα για τις δομές**. Για να ορίσουμε αντικείμενα x,y του τύπου s θα δώσουμε εντολή της μορφής: sx,y; Μετά τον ορισμό των χ, y επιτρέπονται οι εντολές:

x.k =100; y.k=200; x.show (); y.show ();

Οι δύο πρώτες δίνουν τιμή στο μέλος k κάθε αντικειμένου και οι δύο επόμενες εμφανίζουν την τιμή του k για κάθε αντικείμενο.

Στην 3η στήλη

Μαζί με τον τύπο c ορίζεται και το αντικείμενο r. Το αντικείμενο r έχει τρία μέλη. Ένα ακέραιο k μέλος **ιδιωτικό** (***private* –κατά τα καθιερωμένα στις κλάσεις**) και δύο **δημόσια** μέλη συναρτήσεις put () και show () (**επειδή υπάρχει η δήλωση *publc*:**). Η συνάρτηση put() είναι inline ενώ η συνάρτηση show () όχι. Λόγω των αδειών προσπέλασης των μελών δεν επιτρέπεται να χρησιμοποιήσουμε προσπέλαση των αδειών προσπέλασης της μορφής r.k, αλλά μπορούμε να προσπελάσουμε τα μέλη συναρτήσεις. Το μέλος συνάρτησης put δέχεται παράμετρο ένα ακέραιο και τον τοποθετεί σαν τιμή του μέλους k. Το μέλος συνάρτηση show εμφανίζει την τιμή του μέλους k. Έτσι μπορούμε να χρησιμοποιήσουμε τις εντολές:
r.put (100); και r.show (); //η πρώτη θέτει στο k το 100 η δεύτερη το εμφανίζει.

8 Κατασκευαστές και Καταστροφέας

8.1 Συναρτήσεις Κατασκευαστές (constructors)

8.1.1 Ο Καθιερωμένος Κατασκευαστής (default constructor)

Γνωρίζουμε ότι ορίζοντας σε ένα πρόγραμμα μια κλάσης (γενικότερα δομή ή ένωση) μπορούμε στη συνέχεια να δημιουργήσουμε αντικείμενα αυτού του τύπου. Η δημιουργία των αντικειμένων πραγματοποιείται μέσω μιας συνάρτησης που προσφέρεται αυτόματα από την C++ με τον ορισμό της κλάσης και ονομάζεται καθιερωμένος κατασκευαστής (defaultconstructor). Η καθιερωμένη συνάρτηση κατασκευής (defaultconstructor), δημιουργεί ένα αντικείμενο (object), αφιερώνει χώρο μνήμης για το αντικείμενο αλλά δεν τοποθετεί συγκεκριμένες τιμές στα μέλη-δεδομένα του αντικειμένου. Έτσι αν μετά την δημιουργία του αντικειμένου ζητήσουμε εμφάνιση των τιμών των μελών δεδομένων του αντικειμένου θα πάρουμε απροσδιόριστες τιμές. Για να τοποθετήσουμε συγκεκριμένες τιμές στα δεδομένα μέλη ενός αντικειμένου μετά την δημιουργία του μπορούμε να χρησιμοποιήσουμε:

- Εντολές εκχώρησης (assignment) τιμών εάν το δεδομένο μέλος έχει δυνατότητα δημόσιας προσπέλασης (είναι ***publc***) ή
- Δημόσιες συναρτήσεις μέλη που έχουν οριστεί στην κλάση για τον σκοπό αυτό.

8.1.2 Δημιουργία Κατασκευαστή (constructor)

Η C++ μας παρέχει τη δυνατότητα να δημιουργούμε (ορίζοντας μια κλάση) δικές μας συναρτήσεις κατασκευαστές (constructors) οι οποίες αντικαθιστούν την καθιερωμένη συνάρτηση κατασκευαστή (defaultconstructor) κάθε φορά που δημιουργούνται αντικείμενα στο πρόγραμμά μας. Η δημιουργία μιας συνάρτησης κατασκευαστή (constructor) γίνεται με ορισμό συνάρτησης μέλους μέσω εντολής, που έχει την μορφή:

όνομα_κλάσης (λίστα παραμέτρων) {…εντολές…;)

Στον ορισμό της συνάρτησης κατασκευαστή δεν χρησιμοποιούμε τύπο συνάρτησης και η συνάρτηση πρέπει να είναι ***publc***. Η λίστα παραμέτρων καθορίζει τις αναγκαίες πληροφορίες που θα επισυνάπτουμε στα ονόματα των αντικειμένων (που δημιουργούμε με τον constructor), και οι εντολές αναφέρονται στο τι θα εκτελείται κατά την δημιουργία του αντικειμένου (συνήθως εκχωρήσεις τιμών των παραμέτρων στα μέλη του αντικειμένου).

8.1.3 Λόγοι που επιβάλλουν δημιουργία κατασκευαστή

Ο βασικός λόγος που επιβάλλει την δημιουργία ενός κατασκευαστή (constructor) είναι η εκτέλεση κάποιων εντολών (εκτός από τις καθιερωμένες), κατά τον χρόνο που δημιουργούνται τα αντικείμενα. Για παράδειγμα όταν κατά την δημιουργία ενός αντικειμένου θέλουμε:

α)να εκτυπώνονται μηνύματα ή
β)να δίνονται αρχικές τιμές στις μεταβλητές, που είναι μέλη του αντικειμένου (ή και να τροποποιούνται τιμές μεταβλητών μη μελών) ή
γ)να εκτελούνται συναρτήσεις (μέλη του αντικειμένου ή και μη μέλη)

τότε πρέπει να ορίσουμε δικό μας κατασκευαστή (constructor).

8.1.4 Πότε καλείται ένας κατασκευαστής (constructor)

Ο κατασκευαστής (constructor) που έχουμε ορίσει σε μια κλάση, χρησιμοποιείται σε 2 περιπτώσεις:

- Για να ορίσουμε ένα νέο αντικείμενο και
- Για να επαναπροσδιορίσουμε ένα ήδη ορισμένο αντικείμενο

Ο ορισμός ενός νέου αντικειμένου γίνεται με τους εξής τρόπους:
α)όνομα_κλάσης όνομα_αντικειμένου (λίστα_τιμών_παραμέτρων); ή
β)όνομα_κλάσης όνομα_αντικειμένου = όνομα_κλάσης (λίστα_τιμών_παραμέτρων);

Ακόμη σε περίπτωση που η συνάρτηση κατασκευαστής έχει μία μόνο παράμετρο το νέο αντικείμενο μπορεί να οριστεί και ως εξής:
γ)όνομα_κλάσης όνομα_αντικειμένου = τιμή_παραμέτρου;

Σημείωση

Η περίπτωση (γ) είναι μια ειδική περίπτωση και εφαρμόζεται μόνο όταν ο constructor έχει μία μόνο παράμετρο. Αν ο constructor ορίζεται με περισσότερες από μία παραμέτρους δεν εφαρμόζεται.

Ο επαναπροσδιορισμός ενός αντικειμένου γίνεται με τους επόμενους τρόπους:
α)όνομα_αντικειμένου =όνομα_κλάσης (λίστα_τιμών_παραμέτρων) και στην περίπτωση που έχουμε constructor με μια μόνο παράμετρο μπορούμε να επαναπροσδιορίζουμε ένα ήδη ορισμένο αντικείμενο με εντολή της μορφής:
β) όνομα_αντικειμένου = τιμή_παραμέτρου;

8.1.5 Υπερφορτώσεις σε Κατασκευαστές

Στη δημιουργία κλάσεων έχουμε την δυνατότητα να υπερφορτώνουμε (overloading) τις συναρτήσεις κατασκευαστές (constructors) δηλαδή να δημιουργούμε πολλές συναρτήσεις με όνομα το όνομα της κλάσης. Η δυνατότητα αυτή υπόκειται στους γενικούς περιορισμούς υπερφόρτωσης συναρτήσεων δηλαδή πρέπει να έχουμε διαφορετικό πλήθος παραμέτρων ή διαφορές στον τύπο ή διαφορές στην σειρά ως προς την οποία εμφανίζονται οι διάφοροι τύποι των παραμέτρων. Υπερφόρτωση constructor ονομάζεται η δήλωση του ίδιου constructor με διαφορετικά ορίσματα. Αυτό γίνεται όταν θέλουμε ο ίδιος constructor να επιτελεί διαφορετικές λειτουργίες ανάλογα με τις τιμές που λαμβάνει όταν δηλώνουμε αντικείμενα μιας συγκεκριμένης κλάσης. Η δυνατότητα αυτή υπόκειται στους γενικούς περιορισμούς υπερφόρτωσης συναρτήσεων δηλαδή πρέπει να έχουμε διαφορετικό πλήθος παραμέτρων ή διαφορές στον τύπο ή διαφορές στη σειρά με την οποία εμφανίζονται οι διάφοροι τύποι των παραμέτρων.

8.1.6 Καθιερωμένες Παράμετροι σε Κατασκευαστές

Ορίζοντας ένα constructor μπορούμε να δίνουμε στις παραμέτρους του και καθιερωμένες (default) τιμές. Έχοντας ορίσει default παραμέτρους μπορούμε να δημιουργούμε αντικείμενα χρησιμοποιώντας την ίδια συνάρτηση κατασκευαστή με διαφορετικό πλήθος παραμέτρων. Φυσικά και εδώ, όπως και γενικά στις απλές συναρτήσεις, οι καθιερωμένες παράμετροι πρέπει να τοποθετούνται μετά τις κανονικές παραμέτρους (αυτές που δεν παίρνουν default τιμές). Ορίζοντας ένα constructor μπορούμε να έχουμε στις παραμέτρους του και καθιερωμένες (default) παραμέτρους. Έχοντας ορίσει default παραμέτρους μπορούμε να δημιουργούμε αντικείμενα χρησιμοποιώντας την ίδια συνάρτηση κατασκευαστή με διαφορετικό πλήθος παραμέτρων. Φυσικά και εδώ όπως και στις απλές συναρτήσεις οι καθιερωμένες παράμετροι πρέπει να τοποθετούνται μετά τις κανονικές παραμέτρους (αυτές που δεν παίρνουν default τιμές). Επίσης να τονίσουμε ότι είναι εντελώς διαφορετική περίπτωση να έχουμε default παραμέτρους σε constructor από την υπερφόρτωση του constructor, διότι εδώ έχουμε μόνο μια συνάρτηση constructor, ενώ στην υπερφόρτωση του constructor έχουμε πολλές συναρτήσεις constructor.

8.1.7 Αρχικές Τιμές σε Πίνακα Αντικειμένων

Για να δώσουμε αρχικές τιμές σε ένα πίνακα που τα στοιχεία του είναι αντικείμενα μιας κλάσης θα πρέπει να έχουμε ορίσει στην κλάση συνάρτηση κατασκευαστή (constructor). Σημειώνουμε ότι μια συνάρτηση κατασκευαστή με μια παράμετρο χρησιμοποιούμε για δημιουργία αντικειμένων με αρχικές τιμές κατά διαφορετικό τρόπο από εκείνη με δύο ή περισσότερες παραμέτρους. Για παράδειγμα αν έχουμε κλάση data και σε αυτήν έχουμε ορίσει constructor με πρωτότυπο data(***int***k); τότε μπορούμε να ορίσουμε αντικείμενο a με αρχική τιμή το 10 ως εξής: dataa=10; Σε περίπτωση που η συνάρτηση κατασκευαστής έχει δύο ή περισσότερες παραμέτρους π.χ. έχει πρωτότυπο data (***int***k, ***char*** *s); τότε για να ορίσουμε αντικείμενο a με αρχικές τιμές θα πρέπει να δώσουμε δήλωση της μορφής dataa=data (10, "age");

8.2 Συνάρτηση Καταστροφέας (Destructor)

Τα αντικείμενα που δημιουργούνται σε ένα πρόγραμμα καταστρέφονται αυτόματα όταν το πρόγραμμα εκτελέσει όλες τις εντολές του μπλοκ μέσα στο οποίο δημιουργήθησαν τα αντικείμενα (γενικότερα όταν παύσουν τα αντικείμενα να είναι εν ζωή).

Η καταστροφή των αντικείμενων συνεπάγεται την απελευθέρωση του χώρου μνήμης που κατείχαν τα αντικείμενα και δυνατότητα χρησιμοποίησης του χώρου αυτού για υπόλοιπες απαιτήσεις του προγράμματος. Εάν δημιουργούμε στο πρόγραμμά μας αντικείμενα σε δυναμική μνήμη (στο heap) την ευθύνη της καταστροφής των αντικειμένων (δηλαδή την απελευθέρωση του χώρου μνήμης) την έχουμε εμείς και η καταστροφή δεν γίνεται αυτόματα. Φυσικά και στην περίπτωση αυτή ο χώρος μνήμης απελευθερώνεται μετά το πέρας του προγράμματος, αλλά ο χώρος μνήμης μερικές φορές μπορεί να μη επαρκεί κατά την εκτέλεση του προγράμματος επειδή κρατείται για αντικείμενα που πλέον δεν χρειάζονται.

Η C++ μας προσφέρει τη δυνατότητα να δημιουργούμε συνάρτηση καταστροφέα αντικειμένου (destructor) που απελευθερώνει τον χώρο μνήμης του αντικειμένου μόλις το αντικείμενο παύσει να είναι εν ζωή. Η συνάρτηση καταστροφέας (destructor) που δημιουργούμε εκτελεί το έργο της (απελευθέρωση μνήμης) σε κάθε περίπτωση που το αντικείμενο πρέπει να καταστραφεί είτε αυτό δημιουργείτε σε ελεύθερη μνήμη είτε όχι.

8.2.1 Δημιουργία Καταστροφέα (Destructor)

Η συνάρτηση καταστροφέας είναι μια ειδικού τύπου συνάρτηση (όπως ο constructor) δεν έχει τύπο, δεν παίρνει παραμέτρους, το όνομά της είναι το ίδιο με την κλάση (ή ένωση ή δομή) στην οποία ορίζεται και έχει πριν το όνομα του χαρακτήρα '~'. Δηλαδή ένας destructor ορίζεται με εντολές της μορφής: **~ όνομα_κλάσης () {…εντολές…}**

Οι εντολές που τοποθετούμαι στο μπλοκ ορισμού του destructor είναι συνήθως εντολές εκτύπωσης μηνυμάτων ή εντολές απελευθέρωσης δυναμικής μνήμης.

> **Σημείωση**:
> Δεν χρειάζεται να δημιουργήσουμε δικό μας destructor σε περιπτώσεις που δεν κατανέμουμε δυναμική μνήμη σε αντικείμενα. Σε περιπτώσεις που κατανέμουμε δυναμική μνήμη σε αντικείμενα ο καταστροφέας (destructor) είναι απαραίτητος.

Παράδειγμα 1 Εξορισμού (Καθιερωμένες) παράμετροι σε κατασκευαστή

```
#include<iostream.h>
#include<math.h>

class shapes
{
publc:
        shapes(int x1=0,int x2=0,int y1=0,int y2=0)
        {
                if (x1!=0 && x2!=0 && y1!=0 && y2!=0)
                {
                        dist=sqrt(pow((x2-x1),2)+pow((y2-y1),2));
                        cout<<"Distance "<<dist<<endl;
                }

                if (x1!=0 && x2!=0 && y1!=0 && y2==0)
                {
                        area=(x1+x2)*y1/2;
                        cout<<"Area of trapezoid = "<<area<<endl;
                }

                if (x1!=0 && x2!=0 && y1==0 && y2==0)
                {
                        area=x1*x2;
                        cout<<"Area of rectangle = "<<area<<endl;
                }

                if (x1!=0 && x2==0 && y1==0 && y2==0)
                {
                        area=x1*x1;
                        cout<<"Area of square = "<<area<<endl;
                }

                if (x1==0 && x2==0 && y1==0 && y2==0)
                {
                        int i,j;

                        for(i=1;i<=5;i++)
                        {
                                for(j=1;j<=5-i;j++)
                                        cout<<"*";
```

```
                        cout<<endl;
                    }
                }
        }

private:
        double dist;
        int area;
};

voidmain()
{
        int x1,y1,x2,y2,z,ch;

        cout<<"1.Trapezoid."<<endl;
        cout<<"2.Rectangle."<<endl;
        cout<<"3.Square."<<endl;
        cout<<"4.Distans."<<endl;
        cout<<"5.Draw shape."<<endl;

        cout<<"Give your choice: ";
        cin>>ch;

        if (ch==1)
        {
                cout<<"Give base1 and base2 and height of trapezoid: ";
                cin>>x1>>y1>>z;
                shapes s(x1,y1,z);
        }

        if (ch==2)
        {
                cout<<"Give base1 and base2 of rectangle: ";
                cin>>x1>>y1;
                shapes s(x1,y1);
        }

        if (ch==3)
        {
                cout<<"Give base1 of square: ";
                cin>>x1;
                shapes s(x1);
        }

        if (ch==4)
        {
                cout<<"Give coordinates of 1st Point: ";
                cin>>x1>>y1;
                cout<<"Give coordinates of 2nd Point: ";
                cin>>x2>>y2;
```

```
            shapes s(x1,x2,y1,y2);
        }

        if (ch==5)
            shapess;
}
```

9 Κληρονομικότητα

9.1 Απλή Κληρονομικότητα (inheritance)

Ένα από τα δυναμικότερα χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού είναι η κληρονομικότητα (inheritance). Τα αντικείμενα μιας κλάσης μπορεί να κληρονομούν μέλη δεδομένα και συναρτήσεις μιας άλλης κλάσης. Έτσι τις δυνατότητες των αντικειμένων μιας κλάσης μπορούμε να τις επαυξάνουμε δημιουργώντας κλάσεις με επιπρόσθετα επιθυμητά μέλη δεδομένα και συναρτήσεις. Η δημιουργία νέων αντικειμένων μπορεί να στηριχθεί σε ιδιότητες κλάσεων που ήδη έχουν δημιουργηθεί, έχουν ελεγχθεί χρησιμοποιούνται και προσφέρονται σε βιβλιοθήκες στην αγορά λογισμικού.

9.2 Βάση, Παραγόμενη και Άδειες Προσπέλασης

Η κλάση (γενικότερα ένας σύνθετος τύπος-δομή, κλάση, ένωση) που κληρονομείται σε κάποια άλλη λέγεται βάση (base), ενώ εκείνη που κληρονομεί ονομάζεται παραγόμενη (derived) κλάση. Όπως έχει αναφερθεί τα μέλη μιας κλάσης χαρακτηρίζονται με άδειες προσπέλασης ιδιωτικά (***private***:), προστατευμένα (***protected***:) και δημόσια (***publc***:).

Από την παραγόμενη κλάση επιτρέπεται η προσπέλαση των μελών της βάσης τα οποία είναι προστατευμένα (***protected***) ή δημόσια (***publc***). Τα ιδιωτικά (***private***) μέλη της βάσης είναι απροσπέλαστα από την παραγόμενη κλάση. Τα ιδιωτικά (***private***) μέλη μιας κλάσης χαρακτηρίζονται από το ότι, δεν μπορούν να προσπελαστούν παρά μόνο μέσα στον ορισμό συναρτήσεων, που είναι μέλη της κλάσης ή είναι φιλικές προς την κλάση. Τα προστατευμένα μέλη έχουν αυτή την ιδιότητα με μια εξαίρεση :"Μπορεί να προσπελαστούν και μέσα σε ορισμούς συναρτήσεων σε κλάσεις που είναι παραγόμενες". Εάν λοιπόν θέλουμε μια κλάση να αποκρύπτει ένα μέλος έξω από τον ορισμό της, θα δηλώσουμε αυτό το μέλος ιδιωτικό (***private***). Εάν όμως θέλουμε το μέλος ν α μπορεί να προσπελαύνεται και στον ορισμό (και μόνο) παραγόμενων από αυτήν την κλάσεων θα δηλώσουμε το μέλος προστατευμένο (***protected***). **Τόσο τα ιδιωτικά μέλη, όσο και τα προστατευμένα δεν προσπελαύνονται εκτός του ορισμού των κλάσεων (δηλαδή από τα αντικείμενα των κλάσεων). Τα δημόσια μέλη μιας κλάσης είναι προσπελάσιμα μέσα και έξω από τον ορισμό μιας κλάσης. Φυσικά μπορεί να προσπελαστούν και μέσα στον ορισμό μιας παραγόμενης κλάσης**.

9.3 Χαρακτηρισμοί Κληρονομικότητας

Μια κλάση γίνεται παραγόμενη κάποιας άλλης (της βάσης) με τρεις διαφορετικούς χαρακτηρισμούς κληρονομιάς: Η κληρονομιά μπορεί να είναι δημόσια (***publc***) ή προστατευμένη (***protected***) ή ιδιωτική (***private***). Οι χαρακτηρισμοί κληρονομιάς προσδιορίζουν τις άδειες προσπέλασης των μελών που κληρονομούνται από την βάση στην παραγόμενη κλάση.

Εάν η κληρονομιά είναι **δημόσια**, τότε τα μέλη της βάσης που είναι:

α)**δημόσια**, θα είναι για την παραγόμενη κλάση **δημόσια μέλη**

β)**προστατευμένα**, θα είναι για την παραγόμενη κλάση **προστατευμένα μέλη**

γ)**ιδιωτικά**, θα είναι στην παραγόμενη κλάση **απροσπέλαστα**.

Εάν η κληρονομιά είναι **προστατευμένη**, τότε τα μέλη της βάσης, που είναι:

α)**δημόσια**, θα είναι για την παραγόμενη κλάση **προστατευμένα μέλη**

β)**προστατευμένα**, θα είναι για την παραγόμενη κλάση **προστατευμένα μέλη**

γ)**ιδιωτικά**, θα είναι στην παραγόμενη κλάση **απροσπέλαστα**.

Εάν η κληρονομιά είναι **ιδιωτική**, τότε τα μέλη της βάσης, που είναι:

α)**δημόσια**, θα είναι για την παραγόμενη κλάση **ιδιωτικά μέλη**

β)**προστατευμένα**, θα είναι για την παραγόμενη κλάση **ιδιωτικά μέλη**

γ)**ιδιωτικά**, θα είναι στην παραγόμενη κλάση **απροσπέλαστα**.

Ο επόμενος πίνακας δίνει μια σύνοψη του πως τροποποιούνται οι άδειες προσπέλασης των μελών μιας βάσης σε παραγόμενη κλάση ανάλογα με τον χαρακτηρισμό της κληρονομιάς. Ο επόμενος πίνακας δίνει μια σύνοψη του πως τροποποιούνται οι άδειες προσπέλασης των μελών μιας βάσης σε παραγόμενη κλάση, ανάλογα με τον χαρακτηρισμό της κληρονομιάς.

Σε παραγόμενη κλάση με κληρονομιά:	**Public: μέλος (βάσης γίνεται:)**	**Protected: μέλος (βάσης γίνεται:)**	**Private: μέλος (βάσης γίνεται:)**
publc	***publc***:	***protected***:	απροσπέλαστο
protected	***protected***:	***protected***	απροσπέλαστο
private	***private***:	***private***	απροσπέλαστο

9.4 Γιατί και πως χρησιμοποιείται η Κληρονομικότητα

Κληρονομικότητα χρησιμοποιούμε βασικά όταν θέλουμε να τροποποιήσουμε τον κώδικα ενός σύνθετου τύπου. Η δυνατότητα αυτή μας επιτρέπει και "κτίσιμο" καλύτερα δομημένου προγράμματος, διότι από μικρούς σύνθετους τύπους χωρίς να ενδιαφερόμαστε για λεπτομέρειες δημιουργούμε πιο δυναμικούς σύνθετους τύπους.

Παράδειγμα 1 - Πρόγραμμα με κληρονομικότητα τύπου Public

```
#include<iostream.h>
#include<stdlib.h>
```

```
class shapes//Αρχική κλάση ή κλάση βάσης.
{
pu1blic:
        shapes()
        {
            cout<<"constructor of shapes called"<<endl;
            cout<<"give values of length,width,height "<<endl;
            cin>>length>>width>>height;
        }

protected://Ολα τα μέλh μιας κλάσης που δηλώνονται ως protected μπορούν να προσπελασθούν είτε μέσα στην κλάση στην οποία ορίζονται είτε στις παραγόμενες κλάσεις και πουθενά αλλού.
        double length,width,height,area;

        void print_area()
        {
            cout<<"the area is "<<area<<endl;
        }
};

class square:publc shapes //Αυτόσημαίνειότιηκλάση square κληρονομείταιαπότηνκλάση shapes μεκληρονομικότητατύπουpublc.Αυτόσημαίνειοτι:τα publcμέλητης shapes κληρονομούνταιωςpublc, τα protectedως protectedκαιτα privateκαθόλου
{
publc:
        square()
        {
            cout<<"constructor of square called"<<endl;
        }

        void set_area_sq()
        {
            area=length*length;//η length καιη area κληρονομούνταιαπότην shapes
        }

        void print_area_sq()
        {
            print_area();//κληρονομείταιαπότην shapes
        }
};

class rectangle:publc shapes //Αυτόσημάινειοτιηκλάση rectangle κληρονομείταιαπότηνκλάση shapes μεκληρονομικότητατύπουpublc.Αυτόσημαίνειοτι:τα publcμέλητης shapes κληρονομούνταιωςpublc, τα protectedως protectedκαιτα privateκαθόλου
{
        publc:
        rectangle()
        {
```

```
		cout<<"constructor of rectangle called"<<endl;
	}

	void set_area_rec()
	{
		area=length*width;//η length, η width καιη area κληρονομούνταιαπότην shapes
	}

	void print_area_rec()
	{
		print_area();//κληρονομέιταιαπότη shpes
	}
};

class trapezoid:publc shapes //Αυτόσημαίνειότιηκλάση trapezoid κληρονομείταιαπότηνκλά-ση shapes μεκληρονομικότητατύπουpublc.Αυτόσημαίνειότι:ταpublcμέλητης shapes κληρονομούνταιωςpublc, ταprotectedωςprotectedκαιταprivateκαθόλου
{
	publc:
	trapezoid()
	{
		cout<<"constructor of trapezoid called"<<endl;
	}

	void set_area_trap()
	{
		area=(length*width)*height/2;//η length, η width, η height καιη area κληρονομούνταιαπότην shapes
	}

	void print_area_trap()
	{
		print_area();//κληρονομείταιαπότη shapes
	}
};

voidmain()
{
	int ch;
	char ans;

	do
	{
	cout<<"1-area of square"<<endl;
	cout<<"2-area of rectangle "<<endl;
	cout<<"3-area of trapezoid "<<endl;
	cout<<"4-exit "<<endl;

	cout<<"choose "<<endl;
```

```
        cin>>ch;

        switch(ch)
        {
        case 1:{
                                square s //Καλείται αυτομάτως ο constructor της κλάσης
square. Επειδή όμως η square κληρονομείται από τη shapes θα κληθεί αυτομάτως και ο
constructor της shapes
                                s.set_area_sq();
                                s.print_area_sq();
                                break;
                  }

        case 2: {
                                rectangle r;
                                r.set_area_rec();
                                r.print_area_rec();
                                break;
                        }

        case 3:{        trapezoid t;
                                t.set_area_trap();
                                t.print_area_trap();
                                break;
                  }

        case 4: exit(1);
        }

        cout<<"continue? (y/n) "<<endl;
        cin>>ans;
        system("cls");
        }
        while (ans='y');
}
```

Παράδειγμα 2 - Πρόγραμμα με προστατευμένη (protected) κληρονομικότητα

```
#include<iostream.h>
//Β! τρόπος κληρονομικότητας: Προστατευμένη Κληρονομικότητα
//Τα public μέλη κληρονομούνται σαν protected, τα protected σαν protected και τα private
σαν απροσπέλαστα
class A
{
        int x;//Το μέλος x εννοείται ότι είναι private

protected: int y;

publc:  int z;
```

```
        void get(){cout<<" Δώσε x,y,z :";cin>>x>>y>>z;}
        void show(){cout<<"x="<<x<<" y="<<y<<" z="<<z<<'\n';}
};

class B:protected A{};//Η κλάση Β δεν έχει δικά της μέλη, αλλά κληρονομεί τα μέλη της κλάσης Α

voidmain()
{
B b;//Ορίζεται αντικείμενο της κλάσης Β
//b.get(); αυτό είναι λάθος διότι το αντικείμενο b δεν μπορεί να προσπελάσει τη συνάρτηση get() αφού αυτή κληρονομείται σαν protected από τη βάση
//b.show(); αυτό είναι λάθος διότι το αντικείμενο b δεν μπορεί να προσπελάσει τη συνάρτηση show() αφού αυτή κληρονομείται σαν protected από τη βάση
//b.z=2; αυτό είναι λάθος διότι το αντικείμενο b δεν μπορεί να προσπελάσει τη μεταβλητή z αφού αυτή κληρονομείται σαν protected από τη βάση
    //cout<<b.z<<endl; αυτό είναι λάθος διότι το αντικείμενο b δεν μπορεί να προσπελάσει τη μεταβλητή z αφού αυτή κληρονομείται σαν protected από τη βάση
    //b.y=1; αυτό είναι λάθος διότι το y είναι protected μέλος της βάσης άρα δεν μπορεί να προσπελαστεί έξω από τον ορισμό της κλάσης βάσης ή της παραγόμενης κλάσης
    //b.x=0; αυτό είναι λάθος διότι το x είναι private μέλος της βάσης και κληρονομείται σαν απροσπέλαστο
    //Κάθε παραγόμενη κλάση με βάση την κλάση Β δεν έχει προσπέλαση στο μέλος x

//Δηλαδή ουσιαστικά το πρόγραμμα αυτό δεν κάνει τίποτα

//Στην προστατευμένη κληρονομικότητα στην παραγόμενη κλάση δεν μπορούν να προσπελαστούν τα publc και τα protected μέλη της βάσης έξω από την παραγόμενη κλάση αλλά ούτε και τα private μέλη της βάσης

//Τα private μέλη μιας κλάσης προσπελαύνονται μόνο στον ορισμό της κλάσης και πουθενά αλλού
//Τα protected μέλη μιας κλάσης προσπελαύνονται και στον ορισμό της κλάσης αλλά και στον ορισμό των παραγόμενων κλάσεων
//Τα publc μέλη μιας κλάσης είναι προσπελάσιμα και στον ορισμό της κλάσης αλλά και έξω από τον ορισμό της κλάσης δηλαδή παντού
}
```

9.5 Κατασκευαστές και Καταστροφέας σε Κληρονομιά

Δημιουργώντας ένα αντικείμενο μιας παραγόμενης (derived) κλάσης, τότε λειτουργεί αρχικά ο κατασκευαστής (constructor) που είναι προσδιορισμένος για την κλάση βάση και στη συνέχεια ο κατασκευαστής, που είναι προσδιορισμένος για την παραγόμενη κλάση. Αν δεν έχουμε δικούς μας κατασκευαστές, τότε λειτουργούν οι καθιερωμένοι (default).

Όταν καταστρέφεται ένα αντικείμενο μιας παραγόμενης κλάσης, τότε λειτουργεί αρχικά ο καταστροφέας (destructor) της παραγόμενης κλάσης και στη συνέχεια ο καταστροφέας της βάσης (και πάλι αν δεν έχουμε ορίσει δικούς μας καταστροφείς λειτουργούν οι καθιερωμένοι).

Παράδειγμα 1 - Constructor & Destructor σε κληρονομικότητα

```
#include<iostream.h>
```

```
class A
{
protected:
        A()
        {
                cout<<"Object of class A"<<endl;
        }
        ~A()
        {
                cout<<"Destructor of object of class B"<<endl;
        }
};


class B:publc A
{
publc:
        B()
        {

                cout<<"Object of class B"<<endl;
        }
        ~B()
        {
                cout<<"Destructor of object of class B"<<endl;
        }
};

voidmain()
{
        Bb;
}
```

10 Υπερφόρτωση (overloading) Τελεστών

Οι τελεστές στη C++ είναι σύμβολα που αντιπροσωπεύουν ειδικές συναρτήσεις οι οποίες ονομάζονται **συναρτήσεις τελεστές** (operatorfunctions). Χρησιμοποιώντας τον τελεστή + μπορούμε να προσθέσουμε δύο μεταβλητές a, b με την παράσταση a+b. Στην πραγματικότητα καλούμε την συνάρτηση που αντιστοιχίζεται στον τελεστή + με παραμέτρους τους τελεστέους (operands) a, b. Γνωρίζουμε ότι την ίδια παράσταση a+b χρησιμοποιούμε για την πρόσθεση είτε οι τελεστέοι είναι μεταβλητές τύπου ***int*** είτε είναι τύπου ***double***. Αυτό συμβαίνει διότι στη συνάρτηση τελεστή + έχουν γίνει κατάλληλες υπερφορτώσεις (overloadings). Στη C++ έχουμε δυνατότητα να χρησιμοποιούμε τελεστές σε αντικείμενα ενός σύνθετου τύπου αρκεί να υπερφορτώσουμε (overload) τις αντίστοιχες συναρτήσεις τελεστές. Για παράδειγμα επιτρέπεται να υπερφορτώσουμε τον τελεστή + σε μια κλάση ώστε να εκτελεί την πρόθεση δύο αντικειμένων της κλάσης. Το αποτέλεσμα της πράξης το προκαθορίζουμε ανάλογα με τις εντολές που δίνουμε στην υπερφόρτωση της συνάρτησης τελεστή +. Τις συναρτήσεις τελεστές που υπερφορτώνουμε για αντικείμενα κάποιου σύνθετου τύπου (δομή, ένωση κλάση) τις χειριζόμαστε σαν συναρτήσεις που έχουν την μορφή:

τύπος operator σύμβολο (παράμετροι)

Για παράδειγμα αν έχουμε δύο αντικείμενα a, b της κλάσης A και θέλουμε η παράσταση a+b να υπολογίζει ένα ακέραιο που προκύπτει από κάποιες πράξεις σε μέλη των αντικειμένων αρκεί να προσδιορίσουμε μια συνάρτηση τελεστή της οποίας το πρωτότυπο θα έχει την μορφή:

int **operator + (A a, A b);**

10.1 Τρόποι Υπερφόρτωσης των Τελεστών

Η υπερφόρτωση των τελεστών για τα αντικείμενα ενός σύνθετου τύπου μπορεί να γίνει κατά κανόνα με δύο τρόπους:

1. **Ορίζοντας συναρτήσεις τελεστές μέλη του τύπου**
2. **Ορίζοντας φιλικές προς τον τύπο συναρτήσεις τελεστές.**

Οι τελεστές που μπορεί να υπερφορτωθούν έχουν ένα ή δύο τελεστέους και ανάλογα με την περίπτωση θα πρέπει να ορισθούν συναρτήσεις τελεστές με προσδιορισμό μιας ή δύο παραμέτρων [εξαίρεση παρουσιάζεται στον τελεστή ()]. Για παράδειγμα ο τελεστής + έχει δύο τελεστέους και ο τελεστής ++ έχει ένα τελεστέο. Επομένως η υπερφόρτωση του τελεστή + θα πρέπει να γίνει με προσδιορισμό δύο παραμέτρων ενώ η υπερφόρτωση του τελεστή ++ θα πρέπει να γίνει με προσδιορισμό μιας παραμέτρου. Εάν για την υπερφόρτωση τελεστή χρησιμοποιήσουμε συνάρτηση τελεστή μέλος θα πρέπει να έχουμε υπόψη ότι η πρώτη παράμετρος είναι ήδη προσδιορισμένη και δεν πρέπει να αναφερθεί στον ορισμό. Θεωρείται το αντικείμενο που είναι αριστερά του τελεστή (αν έχει δύο τελεστέους) ή το αντικείμενο επί του οποίου ορίζεται η πράξη (αν έχει ένα τελεστέο) **είναι το αντικείμενο *this**. Έτσι χρησιμοποιώντας υπερφόρτωση με συνάρτηση τελεστή μέλος το πρωτότυπο για την υπερφόρτωση του τελεστή + θα έχει μόνο μία παράμετρο (την δεύτερη) δηλαδή θα έχει την μορφή:

τύπος operator + (παράμετρος_δεξιά);

ενώ το πρωτότυπο για την υπερφόρτωση του ++ δεν θα έχει παράμετρο δηλαδή θα έχει την μορφή:

τύπος operator ++();

Χρησιμοποιώντας φιλική συνάρτηση για την υπερφόρτωση θα πρέπει να δώσουμε αντίστοιχα για τον τελεστή + δύο παραμέτρους (η σειρά δεν παίζει ρόλο), δηλαδή θα έχουμε την μορφή:

τύπος operator +($1^{η}$_παράμετρος, $2^{η}$_παράμετρος);

και για τον τελεστή ++ θα πρέπει να δώσουμε μια παράμετρο με την μορφή:
τύπος operator ++(παράμετρος);

10.2 Τελεστές που Υπερφορτώνονται και Περιορισμοί

Η C++ μας επιτρέπει να υπερφορτώσουμε όλους τους γνωστούς τελεστές εκτός των επόμενων 5 τελεστών:

- **::**(τελεστής εύρους)
- **.**(τελεία-προσπέλαση μέλους)
- **.***(τελεία_αστέρι-προσπέλαση μέλους με pointer προς το μέλος)
- **?**(τριαδικός τελεστής)
- **sizeof** (υπολογίζει πλήθος bytes).

Κατά την υπερφόρτωση των τελεστών πρέπει να λάβουμε υπόψη μας τα ακόλουθα:

- **Δεν είναι δυνατόν να δημιουργήσουμε τελεστές με δικά μας σύμβολα**. Το σύμβολο @ δεν είναι ορισμένος τελεστής στη C++ άρα δεν μπορούμε να χρησιμοποιήσουμε τέτοιο τελεστή. Ειδικά σε αντικείμενα μπορούμε να χρησιμοποιήσουμε τον τελεστή ^ υπερφορτώνοντας την αντίστοιχη συνάρτηση τελεστή αφού είναι ορισμένος στην C++.
- Οι υπερφορτωμένοι τελεστές **διατηρούν την βαθμίδα προτεραιότητας** που έχουν και δεν υπάρχει τρόπος να τη μεταβάλλουμε (η πράξη /όπως και να ορίζεται για τα αντικείμενα σε μια παράσταση χωρίς παρενθέσεις θα γίνεται πριν την πράξη +)
- Δεν επιτρέπεται να **αλλάξουμε το πλήθος των τελεστέων** (operands). (ο τελεστής ! ανεξάρτητα του τι θα ορίσουμε να κάνει, θα δέχεται ένα τελεστέο και ο τελεστής +πάντοτε δύο). Εξαίρεση έχουμε στον τελεστή ().
- Στην υπερφόρτωση της συνάρτησης τελεστή **δεν επιτρέπονται καθιερωμένες (default) παράμετροι** (π.χ. όχι δεκτό ***int***operator + (Aa, ***int***k=0))

Οι επόμενοι 4 τελεστές υπερφορτώνονται μόνο με συναρτήσεις μέλη (Στον τελεστή () δεν έχουμε περιορισμό για το πλήθος των παραμέτρων)

- [] (υπολογισμός στοιχείου πίνακα),
- () (υπολογισμός τιμής συνάρτησης),
- .(προσπέλαση μέλους με pointer) και
- =(εκχώρηση τιμής assignment).

Γενικά υπερφορτώνοντας ένα τελεστή προκειμένου να τον χρησιμοποιήσουμε σε αντικείμενα μπορούμε να προσδιορίσουμε ελεύθερα την πράξη που θα επιτελέσει. Για παράδειγμα να χρησιμοποιούμε το σύμβολο + για να εκτελούμε αφαίρεση ή να παίρνουμε μόνιμα το αποτέλεσμα 500. Όμως η υπερφόρτωση των τελεστών γίνεται με στόχο να απλοποιούμε τις παραστάσεις στο πρόγραμμα μας και όχι να περιπλέξουμε το πρόγραμμα. Έτσι θα πρέπει να υπερφορτώνουμε τους τελεστές διατηρώντας όσο το δυνατόν μια λογικά παραδεκτή συμπεριφορά. Η υπερφόρτωση ορισμένων τελεστών σε αντικείμενα θα πρέπει να αποφεύγεται με την έννοια ότι η τροποποίηση δεν επιβάλλεται από ουσιαστικές ανάγκες προγραμματισμού

εκτός από την δημιουργία εκτυπώσεων π.χ δυσκολεύεται κανείς να συλλάβει παράδειγμα που επιβάλλει την υπερφόρτωση του τελεστή &.

10.3 Η Κλάση car ως Παράδειγμα Υπερφόρτωσης

Υποθέτουμε ότι τα αντικείμενα της κλάσης car θα διαθέτουν δύο ιδιωτικά μέλη:
α)τη συμβολοσειρά (string) name 79 το πολύ χαρακτήρων στην οποία καταγράφεται η μάρκα αυτοκινήτου
β)την ακέραια μεταβλητή velocity στην οποία θα τοποθετείται η ταχύτητα του αυτοκινήτου

Ακόμη υποθέτουμε ότι η κλάση car διαθέτει δημόσιες συναρτήσεις μέλη κατασκευαστή (constructor) με τον οποίον μπορούμε να δημιουργούμε αντικείμενα:
α)με αρχική τιμή μόνο τη μάρκα οπότε η ταχύτητα θα παίρνει αυτόματα τιμή μηδέν και
β)με αρχικές τιμές σε μάρκα και σε ταχύτητα

Δηλαδή ο κατασκευαστής θα έχει καθιερωμένη (default) τιμή παραμέτρου για την ταχύτητα και τη συνάρτηση μέλους show() με την οποία θα εμφανίζονται οι τιμές των μελών name και velocity. Είναι γνωστό ότι δημιουργώντας μια κλάση, η C++ μας εφοδιάζει με ένα καθιερωμένο τελεστή =, που μπορούμε να χρησιμοποιήσουμε για τα αντικείμενα της κλάσης. Ο καθιερωμένος τελεστής = είναι δυαδικός τελεστής (δύο τελεστέοι operands) και η ισότητα a=b για τα αντικείμενα a, b προκαλεί την αντιγραφή των μελών δεδομένων του b στα αντίστοιχα μέλη του a.

10.4 Σύνοψη Χρήσης του Συμβόλου = με Αντικείμενα

Το σύμβολο = χρησιμοποιείται σε κλάσεις με δύο τρόπους:
1)Σαν διαχωριστής (separator ή punctuator) κατά τη δημιουργία νέου αντικειμένου με αρχική τιμή άλλο αντικείμενο όπως:
όνομα_κλάσης νέο_αντικείμενο = όνομα_κλάσης (αντικείμενο_υπάρχον);
Στην περίπτωση αυτή διαχωρίζει τον ορισμό του αντικειμένου από την αρχική τιμή.

2)Σαν τελεστής με δύο διακριτές χρήσεις:
α)εκχώρηση τιμής αντικειμένου σε αντικείμενο όπως
αντικείμενο=αντικείμενο; ή αντικείμενο=κλάση (αντικείμενο);
β)εκχώρηση τιμής κάποιου μέλους σε αντικείμενο όπως
αντικείμενο=τιμή; ή αντικείμενο =κλάση (τιμή);

10.5 Υπερφόρτωση των +(πρόσθεση), -(αφαίρεση) +(πρόσημο), -(πρόσημο)

Στην κλάση car (όπως και σε κάθε νέα κλάση) δεν υπάρχουν καθιερωμένοι τελεστές + ή – (όπως ισχύει για τον τελεστή =). Επομένως για τα αντικείμενα της κλάσης a, b οι παραστάσεις a+b; a-b; +a; -a; θα προκαλέσουν σφάλμα εκτός και έχει γίνει υπερφόρτωση των τελεστών + (πρόσθεση και πρόσημο) και –(αφαίρεση και πρόσημο). Θα ορίσουμε υπερφόρτωση των τελεστών αποδίδοντας στους τελεστές πράξεις με τις ακόλουθες έννοιες:

- Με a+b; θα εννοούμε αντικείμενο τύπου car που έχει name μη προσδιορισμένο (ο pointername θα έχει τιμή ***NULL*** (=0)) και velocity το άθροισμα των ταχυτήτων των αντικειμένων a και b (δηλαδή a.velocity+b.velocity)

- Με a-b; θα εννοούμε αντικείμενο τύπου car που έχει name μη προσδιορισμένο (ο pointername θα έχει τιμή ***NULL*** (=0)) και velocity τη διαφορά των ταχυτήτων των αντικειμένων a και b (δηλαδή a.velocity-b.velocity).

· Με +a θα εννοούμε τροποποίηση του αντικειμένου a ως προς την ταχύτητα. Η ταχύτητα θα παίρνει την απόλυτη τιμή της ταχύτητας που είχε (δηλαδή a.velocity=abs(a.velocity)).

· Με –a θα εννοούμε τροποποίηση του αντικειμένου a ως προς την ταχύτητα. Η ταχύτητα θα παίρνει αντίθετη τιμή (δηλαδή a.velocity=-a.velocity).

Η υπερφόρτωση των τελεστών + και - μπορεί να γίνει είτε με συναρτήσεις τελεστές μέλη είτε με φιλικές συναρτήσεις.

10.6 Υπερφόρτωση του Τελεστή <<

Το σύμβολο << ενώ είναι ένας και μοναδικός τελεστής χρησιμοποιείται στην C++ σαν τελεστής σε δύο διαφορετικές περιπτώσεις:

α)σαν ολίσθηση προς τα αριστερά και
β)σαν τελεστής εξόδου προς αντικείμενα που ανήκουν σε προκαθορισμένη από την C++ κλάση με όνομα ostream.

Εάν a, b είναι ακέραιοι τότε με a<<b; εκτελείται ο τελεστής ολίσθησης και παράγει αποτέλεσμα a*2b (b φορές ολίσθηση προς τα αριστερά του δυαδικού αριθμού a).

Αν a είναι αντικείμενο της κλάσης ostream και b μια σταθερά (ή μεταβλητή ή συμβολοσειρά), τότε με a<<b; γίνεται εξαγωγή (output) της τιμής του b υπό μορφή χαρακτήρων και εισαγωγή (insertion) των χαρακτήρων στο αρχείο με το οποίο έχει συνδεθεί το αντικείμενο a. Πράγματι με ***cout***<<15; εκτυπώνεται στην οθόνη το 15, διότι το ***cout*** είναι προκαθορισμένο αντικείμενο της κλάσης ostream και αποτελεί ρεύμα ροής (stream) προς τον στάνταρ μηχανισμό εξόδου (συνήθως οθόνη). Το αποτέλεσμα αυτό προκύπτει, διότι στην κλάση ostream ο τελεστής ολίσθηση έχει υπερφορτωθεί. Ο τελεστής εξόδου << (output) λέγεται και εισαγωγέας (inserter). Δημιουργώντας μια δική μας κλάση δεν έχουμε διαθέσιμο τον τελεστή << ούτε σαν αριστερή ολίσθηση (με πρώτο τελεστέο αντικείμενο της κλάσης και δεύτερο τελεστέο ακέραιο), ούτε σαν τελεστή εξόδου (με πρώτο τελεστέο αντικείμενο της ostream και δεύτερο τελεστέο αντικείμενο της κλάσης που δημιουργούμε). Για να επιτύχουμε κάτι τέτοιο θα πρέπει να υπερφορτώσουμε τον τελεστή <<ολίσθηση, ή τον τελεστή <<έξοδος προς ρεύμα ροής. Η υπερφόρτωση του τελεστή ολίσθησης μπορεί να γίνει, είτε με συνάρτηση μέλος, είτε με συνάρτηση φιλική κατά εντελώς ανάλογο τρόπο με αυτόν που εφαρμόσαμε για τους τελεστές +(άθροισμα) και –(διαφορά) π.χ. στην κλάση car ορίζοντας την υπερφόρτωση μέλους Car&operator<<(***int***k) {velocity=velocity<<k; ***return*** *this;} μπορούμε να χρησιμοποιούμε την παράσταση a<<1; για να διπλασιάζουμε την ταχύτητα στο αντικείμενο a (της κλάσης car). Η έξοδος (output) προς ένα αντικείμενο της κλάσης ostream, όπως είναι το ***cout***, μπορεί να γίνει και πάλι με υπερφόρτωση του τελεστή << (εξόδου – output). Υπάρχουν για τον σκοπό αυτό δύο δυνατότητες:

· να επιχειρήσουμε να υπερφορτώσουμε τον τελεστή <<στην κλάση ostream (και να διαφοροποιήσουμε μια προκαθορισμένη από την C++ κλάση) και
· να υπερφορτώσουμε στην κλάση που δημιουργούμε τον τελεστή εξόδου <<.

Φυσικά είναι αρκετά τολμηρό (έως παράλογο!) να τροποποιούμε τα καθιερωμένα αρχεία της C++ για να εργαστούμε με κάποια κλάση που θα δημιουργήσουμε. Απομένει λοιπόν να υπερφορτώσουμε τον τελεστή εξόδου στην κλάση που δημιουργούμε.

10.7 Υπερφόρτωση του Τελεστή >>

Το σύμβολο >> είναι ένας και μοναδικός τελεστής που χρησιμοποιείται στην C++ σαν τελεστής σε δύο διαφορετικές περιπτώσεις.

α)σαν ολίσθηση προς τα δεξιά και
β)σαν τελεστής εισόδου προς αντικείμενα, που ανήκουν σε προκαθορισμένη από την C++ κλάση με όνομα istream.

Εάν a, b είναι ακέραιοι τότε με a>>b; εκτελείται ο τελεστής ολίσθηση και παράγει αποτέλεσμα a/2β; εκτελείται ο τελεστής ολίσθηση και παράγει αποτέλεσμα α/2b (b φορές ολίσθηση προς τα δεξιά του δυαδικού αριθμού a). Εάν a είναι αντικείμενο της κλάσης istream και b μια μεταβλητή, τότε με a>>b; γίνεται εισαγωγή (input) στο b κάποιας τιμής μετά από εξαγωγή από το αρχείο με το οποίο έχει συνδεθεί το αντικείμενο a. Πράγματι με cin>>b; εισάγουμε από το πληκτρολόγιο τιμή στο b, διότι το cin είναι προκαθορισμένο αντικείμενο της κλάσης istream και αποτελεί ρεύμα ροής (stream) για εξαγωγή από τα στάνταρ μηχανισμού εισόδου (κατά κανόνα το πληκτρολόγιο). Το αποτέλεσμα αυτό προκύπτει, διότι στην κλάση istream ο τελεστής ολίσθηση προς τα δεξιά έχει υπερφορτωθεί. Ο τελεστής εισόδου (input) >> λέγεται και εξαγωγέας (extractor). Δημιουργώντας μια δική μας κλάση δεν έχουμε διαθέσιμο τον τελεστή >> ούτε σαν δεξιά ολίσθηση (με πρώτο τελεστέο αντικείμενο της κλάσης και δεύτερο τελεστέο ακέραιο), αλλά ούτε και σαν τελεστή εισόδου (με πρώτο τελεστέο αντικείμενο της istream) και δεύτερο τελεστέο αντικείμενο της κλάσης που δημιουργούμε). Για να επιτύχουμε κάτι τέτοιο θα πρέπει να υπερφορτώσουμε, είτε τον ένα τελεστή >>ολίσθησης, είτε τον τελεστή >> είσοδος προς ρεύμα ροής. Η υπερφόρτωση του τελεστή ολίσθησης δεξιά μπορεί να γίνει, είτε με συνάρτηση μέλος, είτε με συνάρτηση φιλική κατά εντελώς ανάλογο τρόπο με αυτόν που εφαρμόσαμε για τον τελεστή << π.χ. στην κλάση car ορίζοντας την υπερφόρτωση μέλος: Car&operator>>(***int***k) {velocity = velocity>>k; ***return*** *this;} μπορούμε να χρησιμοποιούμε την παράσταση a>>1; για να υποδιπλασιάζουμε την ταχύτητα στο αντικείμενο a (της κλάσης car). Η είσοδος από ένα αντικείμενο της κλάσης istream όπως είναι το cin, μπορεί να γίνει και πάλι με υπερφόρτωση του τελεστή >> (εισόδου –input). Υπάρχουν για τον σκοπό αυτό δύο δυνατότητες. Πρώτον να επιχειρήσουμε να υπερφορτώσουμε τον τελεστή >> στην κλάση istream (και να διαφοροποιήσουμε μια προκαθορισμένη από την C++ κλάση) και δεύτερον να υπερφορτώσουμε στην κλάση που δημιουργούμε τον τελεστή εισόδου>>. Φυσικά και πάλι η πλέον κατάλληλη διέξοδος είναι να υπερφορτώσουμε τον τελεστή εισόδου >> στην κλάση που δημιουργούμε. Ας υποθέσουμε ότι θέλουμε να επιτύχουμε υπερφόρτωση του τελεστή εισόδου στην κλάση car. Γνωρίζουμε ότι ο πρώτος τελεστέος στο τελεστή εξόδου >> είναι αντικείμενο της κλάσης istream. Επομένως για την υπερφόρτωση δεν μπορούμε να χρησιμοποιήσουμε συνάρτηση μέλος της car διότι τότε ο πρώτος τελεστέος θα ήταν (έμμεσα) το αντικείμενο *this (του τύπου car). Άρα θα χρησιμοποιήσουμε φιλική συνάρτηση. Στην υπερφόρτωση θα εισάγονται δύο παράμετροι ένα αντικείμενο a της istream και ένα αντικείμενο b της car και θα χρησιμοποιείται ο τελεστής εισόδου << για να διαβιβάζονται προς ρεύμα ροής a τα μέλη του αντικειμένου b (μεταβλητές βασικών τύπων). Για να διατηρήσουμε την ιδιότητα πολλαπλής εκτέλεσης του τελεστή >> θα πρέπει η συνάρτηση τελεστής >> να επιστρέφει το αντικείμενο a με ***return***a; Δηλαδή η επιστρεφόμενη τιμή θα είναι μια αναφορά (reference) προς την παράμετρο εισόδου a. Αυτό επιβάλλει ότι και η παράμετρος a θα πρέπει να είναι αναφορά (διότι διαφορετικά θα είχαμε αναφορά προς τοπική μεταβλητή). Από την εκτέλεση του τελεστή >> το αντικείμενο που εκφράζει η παράμετρος b θα πρέπει να τροποποιείται (αφού θα εισάγονται τιμές στα μέλη του). Επομένως και η παράμετρος b θα πρέπει να είναι μια αναφορά (reference) προς αντικείμενα του τύπου car.

10.8 Υπερφόρτωση του Τελεστή ++

Ο τελεστής ++ έχει διπλή συμπεριφορά στη C++. Παρουσιάζεται είτε σαν πρόθεμα (prefix) μιας μεταβλητής (οπότε αυξάνει την τιμή της μεταβλητής πριν επακολουθήσει κάποια πράξη στην οποία συμμετέχει η μεταβλητή) είτε σαν επίθεμα (postfix) (οπότε εκτελείται κάποια πράξη στην οποία συμμετέχει η μεταβλητή και στην συνέχεια αυξάνεται η τιμή της μεταβλητής) π.χ. με b=++a; πρώτα αυξάνεται το a και μετά εκτελείται η ισότητα (εκχωρείται τιμής), ενώ στην παράσταση b=a++; πρώτα γίνεται η εκχώρηση της τιμής του a στο b και μετά αυξάνεται η τιμή της a. Η διπλή αυτή συμπεριφορά του τελεστή ++ οφείλεται στο ότι υπάρχουν αφιερωμένες στον τελεστή δύο συναρτήσεις τελεστές. Η πρώτη αντιστοιχίζεται στην συμπεριφορά σαν πρόθεμα και παίρνει μια παράμετρο (την μεταβλητή που αυξάνει) ενώ η δεύτερη αντιστοιχίζεται στην συμπεριφορά σαν επίθεμα και παίρνει δύο παραμέτρους. Η πρώτη παράμετρος του τελεστή ++ σαν επίθεμα είναι η μεταβλητή που αυξάνει (αριστερά του τελεστή) και η δεύτερη είναι μια μεταβλητή ακέραιου τύπου η οποία δεν χρησιμοποιείται και μπορεί να παραληφθεί. Η δεύτερη παράμετρος στην συνάρτηση τελεστής ++ επίθεμα απλά χρησιμεύει για την διαφοροποίηση από την συνάρτηση τελεστής ++ πρόθεμα.

Στα αντικείμενα μιας κλάσης όπως η car υπερφορτώνοντας τον τελεστή ++ με μια παράμετρο είναι σαν να υπερφορτώνουμε την συμπεριφορά του τελεστή σαν πρόθεμα, ενώ υπερφορτώνοντας τον τελεστή με δύο παραμέτρους προσδιορίζουμε την συμπεριφορά του τελεστή ++ σαν επίθεμα. Οι υπερφορτώσεις του ++ μπορεί να γίνουν είτε με συναρτήσεις μέλη, είτε με φιλικές συναρτήσεις.

Στην υπερφόρτωση σαν πρόθεμα αυξάνουμε την τιμή της πρώτης παραμέτρους και επιστρέφουμε την πρώτη παράμετρο. Επομένως η πρώτη παράμετρος θα είναι αναφορά (reference) προς αντικείμενο και η επιστρεφόμενη τιμή θα είναι το ίδιο το αντικείμενο παράμετρος αφού τροποποιηθεί δηλαδή και πάλι αναφορά προς αντικείμενο.

Στην υπερφόρτωση σαν επίθεμα θα πρέπει να επιστρέψουμε ένα τοπικό αντικείμενο που έχει την τιμή την πρώτη παράμετρο και μετά να αυξήσουμε την τιμή της παραμέτρου. Επομένως μέσα στην συνάρτηση υπερφόρτωσης θα πρέπει να τροποποιήσουμε μόνιμα την πρώτη παράμετρο. Άρα η πρώτη παράμετρος θα είναι αναφορά. Η επιστρεφόμενη τιμή θα είναι το τοπικό αντικείμενο (όχι αναφορά).

10.9 Παραδείγματα υπερφόρτωσης τελεστών

Υπερφόρτωση του τελεστή =

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

class car
{
char name[80];
int velocity;

public:
    car(char s[80],int v=0)
    {
       strcpy(name,s);
       velocity=v;
    }
```

```
    void show()
    {
       cout<<name<<" TAXYTHTA= "<<velocity<<'\n';
    }
};

voidmain()
{
    car a("volvo "),b("Fiat "),c("Renault ",40),d("Mazda ",100);//Ορίζονται τα αντικείμενα a,b,c,d

    a.show();
    b.show();
    c.show();
    d.show(); //Εμφανίζονται τα μέλη των a,b,c,d

    a=car(c); //Είναι το ίδιο με την καταχώριση a=c
    b=a;//Εμφανίζονται τα μέλη των a,b,c,d
    cout<<'\n';

    a.show();
    b.show();
    c.show();
    d.show(); //Εμφανίζονται τα μέλη των a,b,c,d
    cout<<'\n';

    a=b=c=d;//πολλαπλή καταχώριση
    a.show();
    b.show();
    c.show();
    d.show(); //Εμφανίζονται τα μέλη των a,b,c,d
}
```

Υπερφόρτωση του τελεστή = με συνάρτηση μέλος

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

class car
{
char name[80];
int velocity;

public:
    car (char s[80], int v=0) //τίθεται το s στο name και το v στο velocity
    {
       strcpy(name,s);
       velocity=v;
    }
```

```
    void show()
    {
       cout<<name<<" TAXYTHTA ="<<velocity<<'\n';
    }

    car operator=(carb)//υπερφόρτωση συνάρτησης τελεστή =
    {
       velocity=b.velocity;//Με τον τελεστή καταχώρισης = τροποποιείται μόνο η velocity
       return *this;//επιστρέφεται αντίγραφο του τρέχοντος αντικειμένου
    }
};
```

Γ ΜΕΡΟΣ – ΓΛΩΣΣΑ JAVA

1 Εισαγωγήστη Java

Η γλώσσα Java ανήκει στην 5η γενιά και συγκεκριμένα στις αντικειμενοστραφείς γλώσσες. Όταν ζητάμε από τη Java να μεταγλωττίσει το πρόγραμμα μας αυτή δεν παράγει ένα πρόγραμμα όπως αυτά που έχουμε συνηθίσει να βλέπουμε. Αντί να δημιουργήσει ένα εκτελέσιμο αρχείο γεμάτο από εντολές μηχανής, ο μεταγλωττιστής της Java παράγει στην έξοδο αυτό που είναι γνωστό ως κώδικας byte Java (Java byte codes). Ο κώδικας byte Java είναι εντολές γραμμένες για κάποια εικονική μηχανή Java (virtual machine) που δεν υπάρχει πραγματικά. Για να εκτελέσουμε το πρόγραμμά μας, πρέπει να έχουμε ένα ερμηνευτή κώδικα byte java, πράγμα που σημαίνει ότι στην πραγματικότητα εξομοιώνουμε την εικονική μηχανή Java (JVM) σε όποιον υπολογιστή και όποιο λειτουργικό σύστημα χρησιμοποιούμε. Κανονικά δεν θα το αντιλαμβανόμαστε αυτό, αφού η Java το κάνει αυτόματα όταν εκτελούμε την εφαρμογή μας μέσα στο Αλληλεπιδραστικό Περιβάλλον Ανάπτυξης (Interactive Development Environment – IDE) της Java.
Όταν εκτελούμε μια μικροεφαρμογή στον Ιστό (Web), είναι ευθύνη του φυλλομετρητή (browser) να υλοποιήσει τον κώδικα byte της java. Η προσέγγιση μέσω της εικονικής μηχανής Java εξυπηρετεί 3 σκοπούς:

- **Ανεξαρτησία από την μηχανή.** Υπάρχει μία μόνο Εικονική Μηχανή Java. (Στην πραγματικότητα, δεν υπάρχει καμία – αυτό είναι που την κάνει εικονική). Υπάρχει ένας διαφορετικός εξομοιωτής της Εικονικής Μηχανής Java για κάθε διαφορετικό τύπο PC. Αυτό είναι που δίνει στην Java την ανεξαρτησία της από τη μηχανή.

- **Ασφάλεια.** Η πείρα έχει δείξει πόσο δύσκολο είναι να επιτευχθεί ένα λογικό επίπεδο ασφαλείας στο λογισμικό. Από τη στιγμή που τα προγράμματα εκτελούνται στην Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ – CPU) σε εγγενή κατάσταση (nativemode), οποιαδήποτε ατέλεια στο λειτουργικό σύστημα ή στο φυλλομετρητή μπορεί να αξιοποιηθεί από τους χάκερ που ψάχνουν για μια είσοδο στη μηχανή – πελάτη. Τα προγράμματα της Java δεν έχουν αυτή την ευελιξία. Ένα πρόγραμμα Java εκτελείται στην εικονική μηχανή Java που δημιουργείται από το φυλλομετρητή. Καθώς το πρόγραμμα εκτελείται, ο φυλλομετρητής είναι πάντοτε παρών, παρακολουθώντας κατά κάποιον τρόπο το πρόγραμμα ώστε να εξασφαλίσει ότι αυτό δεν θα κάνει κάτι που δεν πρέπει.

- **Μικρότερο μέγεθος μικροεφαρμογών**. Τα προγράμματα της Εικονικής μηχανής Java είναι πολύ μικρότερα από τα συμβατικά προγράμματα. Αυτό συμβαίνει γιατί οι ενσωματωμένες συναρτήσεις βιβλιοθήκης της Java, τις οποίες καλεί το πρόγραμμά μας, βρίσκονται μέσα στο φυλλομετρητή αντί να συνδέονται (link) στο πρόγραμμα. Αυτό μειώνει σημαντικά το μέγεθος των μικροεφαρμογών Java Που πρέπει να "κατέβουν" (download), οπότε μειώνει και το χρόνο μεταφοράς. Η Java μειώνει ακόμη περισσότερο το χρόνο μεταφοράς στο σύστημά μας, κατεβάζοντας μόνον ότι χρειάζεται. (Ο κώδικας byte που παράγεται για μια δεδομένη κλάση δεν μεταφέρεται μέχρι να τον χρειαστεί το πρόγραμμα).

- **Έγκαιρη Μεταγλώττιση.** Υπάρχει ένα μειονέκτημα στην ερμηνεία του κώδικα byte της Java από το φυλλομετρητή – μειωμένη απόδοση. Τα προγράμματα κώδικα byte της Java δεν εκτελούνται τόσο γρήγορα όσο τα κανονικά προγράμματα, εγγενούς κώδικα. Για να αντιμετωπίσει αυτό το πρόβλημα, το MicrosoftInternetExplorer υποστηρίζει μια ευκολία που ονομάζεται Έγκαιρη Μεταγλώττιση (compilationJust-In-Time, ή JIT). Όταν το InternetExplorer δέχεται για πρώτη φορά μια μικροεφαρμογή Java, αν η Έγκαιρη Μεταγλώτ-

τιση είναι ενεργοποιημένη, μετατρέπει τον κώδικα byte της Java σε εγγενές πρόγραμμα, το οποίο και αποθηκεύει στο δίσκο. Στην πράξη, η Έγκαιρη Μεταγλώττιση αντιμετωπίζει τον κώδικα Byte της Java σαν γλώσσα προέλευσης και τον μεταγλωττίζει σε γλώσσα της τοπικής μηχανής. Η Έγκαιρη Μεταγλώττιση διατηρεί πολλά από τα πλεονεκτήματα του κώδικα Byte της Java. Από τη στιγμή που δεν αλλάζει των κώδικα Java που παράγεται από τη Java, το πρόγραμμα που προκύπτει εξακολουθεί να είναι ασφαλές και ανεξάρτητο από τη μηχανή και οι χρόνοι μεταφοράς παραμένουν ελαχιστοποιημένη. Το Έγκαιρα Μεταγλωττισμένο πρόγραμμα εκτελείται πολλές φορές γρηγορότερα, ακόμη και από τους καλύτερους εξομοιωτές κώδικα byteJava.

1.1 Πλεονεκτήματα Java

Όταν ζητάμε από τη Java να μεταγλωττίσει το πρόγραμμα μας αυτή δεν παράγει ένα πρόγραμμα όπως αυτά που έχουμε συνηθίσει να βλέπουμε. Αντί να δημιουργήσει ένα εκτελέσιμο αρχείο γεμάτο από εντολές μηχανής, ο μεταγλωττιστής της Java παράγει στην έξοδο αυτό που είναι γνωστό ως κώδικας byteJava (Javabytecodes). Ο κώδικας byteJava είναι εντολές γραμμένες για κάποια εικονική μηχανή Java (virtualmachine) που δεν υπάρχει πραγματικά. Για να εκτελέσουμε το πρόγραμμά μας, πρέπει να έχουμε ένα ερμηνευτή κώδικα bytejava, πράγμα που σημαίνει ότι στην πραγματικότητα εξομοιώνουμε την εικονική μηχανή Java (JVM) σε όποιον υπολογιστή και όποιο λειτουργικό σύστημα χρησιμοποιούμε. Κανονικά δεν θα το αντιλαμβανόμαστε αυτό, αφού η Java το κάνει αυτόματα όταν εκτελούμε την εφαρμογή μας μέσα στο Αλληλεπιδραστικό Περιβάλλον Ανάπτυξης (InteractiveDevelopmentEnvironment – IDE) της Java.

Όταν εκτελούμε μια μικροεφαρμογή στον Ιστό (Web), είναι ευθύνη του φυλλομετρητή (browser) να υλοποιήσει τον κώδικα byte της java. Η προσέγγιση μέσω της εικονικής μηχανής Java εξυπηρετεί 3 σκοπούς:

- **Ανεξαρτησία από την μηχανή.** Υπάρχει μία μόνο Εικονική Μηχανή Java. (Στην πραγματικότητα, δεν υπάρχει καμία – αυτό είναι που την κάνει εικονική). Υπάρχει ένας διαφορετικός εξομοιωτής της Εικονικής Μηχανής Java για κάθε διαφορετικό τύπο PC. Αυτό είναι που δίνει στην Java την ανεξαρτησία της από τη μηχανή.
- **Ασφάλεια.** Η πείρα έχει δείξει πόσο δύσκολο είναι να επιτευχθεί ένα λογικό επίπεδο ασφαλείας στο λογισμικό. Από τη στιγμή που τα προγράμματα εκτελούνται στην Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ – CPU) σε εγγενή κατάσταση (nativemode), οποιαδήποτε ατέλεια στο λειτουργικό σύστημα ή στο φυλλομετρητή μπορεί να αξιοποιηθεί από τους χάκερ που ψάχνουν για μια είσοδο στη μηχανή – πελάτη. Τα προγράμματα της Java δεν έχουν αυτή την ευελιξία. Ένα πρόγραμμα Java εκτελείται στην εικονική μηχανή Java που δημιουργείται από το φυλλομετρητή. Καθώς το πρόγραμμα εκτελείται, ο φυλλομετρητής είναι πάντοτε παρών, παρακολουθώντας κατά κάποιον τρόπο το πρόγραμμα ώστε να εξασφαλίσει ότι αυτό δεν θα κάνει κάτι που δεν πρέπει.
- **Μικρότερο μέγεθος μικροεφαρμογών**. Τα προγράμματα της Εικονικής μηχανής Java είναι πολύ μικρότερα από τα συμβατικά προγράμματα. Αυτό συμβαίνει γιατί οι ενσωματωμένες συναρτήσεις βιβλιοθήκης της Java, τις οποίες καλεί το πρόγραμμά μας, βρίσκονται μέσα στο φυλλομετρητή αντί να συνδέονται (link) στο πρόγραμμα. Αυτό μειώνει σημαντικά το μέγεθος των μικροεφαρμογών Java Που πρέπει να "κατέβουν" (download), οπότε μειώνει και το χρόνο μεταφοράς. Η Java μειώνει ακόμη περισσότερο το χρόνο μεταφοράς στο σύστημά μας, κατεβάζοντας μόνον ότι χρειάζεται. (Ο κώδικας byte που παράγεται για μια δεδομένη κλάση δεν μεταφέρεται μέχρι να τον χρειαστεί το πρόγραμμα).

- **Έγκαιρη Μεταγλώττιση.** Υπάρχει ένα μειονέκτημα στην ερμηνεία του κώδικα byte της Java από το φυλλομετρητή – μειωμένη απόδοση. Τα προγράμματα κώδικα byte της Java δεν εκτελούνται τόσο γρήγορα όσο τα κανονικά προγράμματα, εγγενούς κώδικα. Για να αντιμετωπίσει αυτό το πρόβλημα, το MicrosoftInternetExplorer υποστηρίζει μια ευκολία που ονομάζεται Έγκαιρη Μεταγλώττιση (compilationJust-In-Time, ή JIT). Όταν το InternetExplorer δέχεται για πρώτη φορά μια μικροεφαρμογή Java, αν η Έγκαιρη Μεταγλώττιση είναι ενεργοποιημένη, μετατρέπει τον κώδικα byte της Java σε εγγενές πρόγραμμα, το οποίο και αποθηκεύει στο δίσκο. Στην πράξη, η Έγκαιρη Μεταγλώττιση αντιμετωπίζει τον κώδικα Byte της Java σαν γλώσσα προέλευσης και τον μεταγλωττίζει σε γλώσσα της τοπικής μηχανής. Η Έγκαιρη Μεταγλώττιση διατηρεί πολλά από τα πλεονεκτήματα του κώδικα Byte της Java. Από τη στιγμή που δεν αλλάζει των κώδικα Java που παράγεται από τη Java, το πρόγραμμα που προκύπτει εξακολουθεί να είναι ασφαλές και ανεξάρτητο από τη μηχανή και οι χρόνοι μεταφοράς παραμένουν ελαχιστοποιημένη. Το Έγκαιρα Μεταγλωττισμένο πρόγραμμα εκτελείται πολλές φορές γρηγορότερα, ακόμη και από τους καλύτερους εξομοιωτές κώδικα byteJava.

1.2 Διαφορές C++ από Java

- Η Java δεν επιτρέπει πολλαπλή κληρονομικότητα (για λόγους απλότητας)
  - Κάθε κλάση μπορεί να κληρονομεί το πολύ από μία άλλη κλάση («extends»)
  - Interfaces: Δομές που περιέχουν «μη υλοποιημένες» συναρτήσεις, χωρίς μεταβλητές-μέλη και δεν είναι κλάσεις
  - Μία κλάση μπορεί να κάνει «implement» περισσότερα από ένα interface
- Η Java θεωρείται εν γένει απλούστερη γλώσσα από τη C++.
- Όλες οι Java μέθοδοι είναι όπως οι virtual της C++.
- Η Java δεν υποστηρίζει δείκτες (pointers).
- Η Java δενυποστηρίζει defines, typedefs ή preprocessor. Οπότε, δε χρειάζεται ούτε αρχεία κεφαλίδας (header files).
- Στη Java δεν υποστηρίζονται καθολικές μεταβλητές. Εναλλακτικά: “static”
- Στη Java δεν επιτρέπονται συναρτήσεις εκτός κλάσεων (stand-alone functions).

1.3 Δομή Προγράμματος Java

Ένα πρόγραμμα σε java έχει την ακόλουθη μορφή:

publicclass όνομα //το όνομα της κλάσης πρέπει να είναι ίδιο με το όνομα του αρχείου

{//έναρξη κλάσης

publicstaticvoidmain(***String*** [] ***args***) //έναρξη κύριου προγράμματος

```
    {
        δήλωση μεταβλητών;

        εντολή 1;
        εντολή 2;
        …….
    εντολή ν;
    } //τέλος κύριου προγράμματος
} //τέλος κλάσης
```

Παράδειγμα

```
public class HelloWorld
{
    public static void main (String [] args)
    {
        System.out.println ("Hello World!");
    }
}
```

Το *HelloWorld* είναι ίσως το πιο απλό πρόγραμμα που μπορεί κανείς να φανταστεί. Παρόλα αυτά μας ενδιαφέρει για πολλούς λόγους. Ας το εξετάσουμε γραμμή προς γραμμή.

Η αρχική δήλωση ***class*** μπορεί να θεωρείται ότι προσδιορίζει το όνομα του προγράμματος, στη συγκεκριμένη περίπτωση HelloWorld. Ocompiler στην πραγματικότητα παίρνει το όνομα από τη δήλωση ***class***HelloWorld στο αρχείο πηγαίου κώδικα. Αν υπάρχουν παραπάνω από μία κλάση σε ένα αρχείο, τότε ο compiler της Java θα αποθηκεύσει το καθένα σ' ένα ξεχωριστό ***.class*** αρχείο. ToHelloWorld***class*** περιλαμβάνει μία μέθοδο, τη ***main***. Όπως και στη C, η μέθοδος ***main*** μας δείχνει από που ξεκινά να εκτελείται μία εφαρμογή. Η μέθοδος δηλώνεται δημόσια (***public***) δηλαδή μπορούν να την καλέσουν από παντού. Δηλώνεται στατική (***static***) δηλαδή όλα τα παραδείγματα της κλάσης μοιράζονται την ίδια μέθοδο. Δηλώνεται κενή (***void***) που σημαίνει ότι αυτή η μέθοδος δεν επιστρέφει τιμή, όπως και στη C.

Όταν καλείται η μέθοδος ***main*** τυπώνει το HelloWorld! στην οθόνη. Αυτό επιτυγχάνεται με μέθοδο ***System.out.println***. Για να είμαστε πιο ακριβείς, αυτό επιτυγχάνεται καλώντας τη συνάρτηση println() του πεδίου out που ανήκει στην κλάση System. Αλλά για την ώρα θα την θεωρούμε σαν μία μέθοδο.

Αντίθετα με την printf στη C η μέθοδος ***System.out.println*** προσθέτει μια καινούρια γραμμή στο τέλος της εξόδου. Έτσι δεν είναι ανάγκη να συμπεριλάβουμε το \n στο τέλος κάθε αλφαριθμητικού.

1.4 Κλάσεις

1.5 Κλάσεις και Αντικείμενα

Ο αντικειμενοστραφής προγραμματισμός (ObjectOrientedProgramming ή εν συντομία OOP) περιλαμβάνει όλα τα χαρακτηριστικά του δομημένου προγραμματισμού καθώς και ισχυρούς τρόπους οργάνωσης αλγορίθμων και δομών δεδομένων. Οι αντικειμενοστραφείς γλώσσες προγραμματισμού έχουν τρία χαρακτηριστικά: **ενθυλάκωση, πολυμορφία ή πολυμορφισμός και κληρονομικότητα**.

Το πρωταρχικό λοιπόν χαρακτηριστικό των αντικειμενοστραφών γλωσσών προγραμματισμού είναι η **κλάση** (***class***). Η κλάση είναι μια δομή δεδομένων που μπορεί να συσχετιστεί με μεθόδους (συναρτήσεις) που ενεργούν πάνω σε ένα αντικείμενο. Στις γλώσσες προγραμματισμού πριν από τον αντικειμενοστραφή προγραμματισμό οι μέθοδοι (συναρτήσεις) και τα δεδομένα ήταν ξεχωριστά. Στις αντικειμενοστραφείς γλώσσες προγραμματισμού όλα αυτά είναι μέρη μιας κλάσης.

Οι γλώσσες προγραμματισμού παρέχουν ορισμένους απλούς τύπους δεδομένων όπως ***int***, ***float*** και ***String***. Όμως, πολύ συχνά τα δεδομένα δεν είναι τόσο απλά όπως ***int***, ***float*** ή ***String***. Οι κλάσεις επιτρέπουν στους προγραμματιστές να ορίσουν δικούς τους, πιο πολύπλοκους τύπους δεδομένων.

Για παράδειγμα ας υποθέσουμε ότι το πρόγραμμα μας χρειάζεται να κρατάει μια βάση δεδομένων ενός WEBsite. Για κάθε site έχουμε ένα όνομα, ένα URL και μια περιγραφή. Στις παραδοσιακές γλώσσες προγραμματισμού θα υπάρχουν τρεις διαφορετικές μεταβλητές ***String*** για κάθε WEBsite. Με μια κλάση συνδυάζουμε όλα αυτά σε ένα όπως παρακάτω:

```
class website
{
   String name;
   String url;
   Stringdescription;
}
```

Αυτές οι μεταβλητές (όνομα, url και περιγραφή) ονομάζονται **μέλη της κλάσης**. Μας λένε τι είναι μια κλάση και ποιες είναι οι ιδιότητες της.

Σε μια βάση δεδομένων με Websites του θα έχουμε αποθηκευμένες πολλές χιλιάδες websites. Κάθε ξεχωριστό Website είναι ένα αντικείμενο. **Ένα αντικείμενο είναι ένα ιδιαίτερο στιγμιότυπο της κλάσης**. Όταν δημιουργούμε ένα νέο αντικείμενο λέμε ότι δημιουργούμε ένα **στιγμιότυπο** της κλάσης. Κάθε κλάση υπάρχει μόνο μια φορά σε ένα πρόγραμμα, αλλά υπάρχουν πολλά αντικείμενα που είναι στιγμιότυπα κλάσης.

Για να δημιουργήσουμε ένα νέο αντικείμενο στη Java χρησιμοποιούμε τηνεντολή***new***. Για παράδειγμα για να δημιουργήσουμε ένα καινούργιο Website γράφουμε την εντολή:

```
websitex = newwebsite();
```

Καθώς έχουμε δημιουργήσει ένα Website θέλουμε να μάθουμε πληροφορίες γιαυτό. Για να πάρουμε τις μεταβλητές μέλους του website μπορούμε να χρησιμοποιήσουμε τον τελεστή **‘.’**. Το website έχει **τρεις μεταβλητές μέλους**: όνομα, url και περιγραφή. Έτσι το x έχει τρεις μεταβλητές μέλους τις x.name, x.url, x.description. Μπορούμε να το χρησιμοποιήσουμε ακριβώς όπως θα χρησιμοποιούσαμε οποιαδήποτε άλλη μεταβλητή ***String***. Για παράδειγμα:

```
websitex = new website();
x.name = "Ειδήσεις-Νέα";
x.url = "http://www.in.gr/";
x.description = "Ενημερωτικό site";
System.out.println (x.name + " at " + x.url + " is " + x.description);
```

Γενικά ένας ορισμός τύπου κλάσης από το χρήστη έχει την εξής μορφή:

```
class MyClass
{
    //τα μέλη της κλάσης μπαίνουν εδώ
}
```

Πριν από τη λέξη-κλειδί ***class*** μπαίνει ο τύπος πρόσβασης. Προς το παρόν, θα υποθέσουμε τον προκαθορισμένο (default) τύπο πρόσβασης. Η λέξη – κλειδί ***class*** ακολουθείται από το όνομα της κλάσης, το οποίο πρέπει να είναι έγκυρο αναγνωστικό της Java. Το οριοθετημένο με άγκιστρα σώμα της κλάσης περιέχει οποιοδήποτε πλήθος ορισμών μελών. Τα μέλη της κλάσης μπορούν να εμφανίζονται με οποιαδήποτε σειρά μέσα στο σώμα της κλάσης. Μια κλάση έχει 2 τύπους μελών: **μέλη δεδομένων** ή **μεταβλητές μέλη** (datamembers) και **συναρτήσεις μέλη** (functionmembers).

1.6 Ορισμός μελών δεδομένων

Τα μέλη δεδομένων (datamembers) χρησιμοποιούνται για να περιγράψουν τις ιδιότητες των δεδομένων της κλάσης. Δηλώνονται με τους ίδιους κανόνες όπως και οι τοπικές μεταβλητές. Ας δούμε την κλάση BankAccount:

```
class BankAccount
{
  //Ορισμός μελών δεδομένων:
  double currentbalance; //το τρέχον υπόλοιπο λογαριασμού
```

```
    int id; //αριθμός λογαριασμού


}
```

Το τρέχον υπόλοιπο λογαριασμού είναι μια σημαντική ιδιότητα ενός τραπεζικού λογαριασμού. Επιπλέον ο αριθμός λογαριασμού είναι αρκετά σημαντικός γιατί ορίζει τον τραπεζικό λογαριασμό με μοναδικό τρόπο. Ο ορισμός του CurrentBalance παρέχει στην κλάση ένα μέρος για να διατηρεί τις πληροφορίες σχετικά με το υπόλοιπο του λογαριασμού.

1.7 Ορισμός συναρτήσεων μελών (μεθόδων)

Τα μέλη μιας κλάσης μπορούν να είναι και συναρτήσεις. Οι **συναρτήσεις μέλη ονομάζονται και μέθοδοι της κλάσης ή απλώς μέθοδοι** (methods). Ο ορισμός μιας συνάρτησης μέλους (memberfunction) μοιάζει με τον ακόλουθο κώδικα:

```
class BankAccount
{
    double currentbalance; //τρέχον υπόλοιπο


    //οι κλάσεις μπορούν να έχουν και συναρτήσεις-μέλη
    void deposit (double amount) //deposit – κατάθεση
    {
        if (amount >= 0.0)
            currentbalance + =amount;
    }


    void withdrawal (double amount) //withdrawal – ανάληψη
    {
        if (amount >=0.0 && amount < currentbalance)
            currentbalance -= amount;
    }
}
```

Ο ορισμός μιας μεθόδου αρχίζει με τον επιστρεφόμενο τύπο δηλαδή τον τύπο του αντικειμένου που επιστρέφει η μέθοδος. **Αν η μέθοδος δεν επιστρέφει αντικείμενο ο επιστρεφόμενος τύπος είναι *void***. Ο επιστρεφόμενος τύπος ακολουθείται από το όνομα της μεθόδου

που μπορεί να είναι οποιοδήποτε έγκυρο αναγνωριστικό της Java. Το όνομα της μεθόδου ακολουθείται από παρενθέσεις που περιέχουν τη λίστα παραμέτρων (parameterlist). Η λίστα παραμέτρων είναι μια οριοθετημένη με κόμματα λίστα δηλώσεων των ορισμάτων της μεθόδου. Αν η μέθοδος δεν δέχεται ορίσματα, η λίστα παραμέτρων πρέπει να είναι κενή.

Κανένα από τα ορίσματα δεν επιτρέπεται να έχει το ίδιο όνομα με μια μεταβλητή που δηλώνεται τοπικά μέσα στη μέθοδο. Έτσι το παρακάτω τμήμα κώδικα παράγει ένα λάθος μεταγλώττισης:

```
void fn(int i, int size)
{
    //το i στον παρακάτω βρόχο ξαναδηλώνεται. Αυτό δεν επιτρέπεται.
    for (int i=0; i<size; i++)
    {
        ……
    }
}
```

Τα άγκιστρα που ακολουθούν τη λίστα παραμέτρων περιέχουν το σώμα της μεθόδου. Όταν η μέθοδος καλείται ο έλεγχος περνάει στο άγκιστρο ανοίγματος της μεθόδου. Ο προγραμματιστής μπορεί να βγει από οποιοδήποτε σημείο της μεθόδου με τη λέξη κλειδί ***return***. Αν η ροή του προγράμματος δεν συναντήσει εντολή ***return*** τότε η έξοδος από τη μέθοδο γίνεται στο άγκιστρο κλεισίματος. Αν η μέθοδος επιστρέφει κάτι διαφορετικό από ***void*** απαιτείται ένα ***return*** ακολουθούμενο από μια παράσταση που δείχνει στην επιστρεφόμενη τιμή. Αυτό φαίνεται στην επόμενη μέθοδο Balance που επιστρέφει το υπόλοιπο του τρέχοντος λογαριασμού:

```
class BankAccount
{
    //τρέχον υπόλοιπο
    double currentbalance;

    double balance()//επιστροφή τρέχοντος υπολοίπου
    {
       return currentbalance;
    }
}
```

Η έξοδος από μια μέθοδο **με επιστρεφόμενο τύπο διαφορετικό από *void*πρέπει να γίνεται μέσω της εντολής *return***γιατί το άγκιστρο κλεισίματος δεν επιστρέφει τιμή.

1.8 Διαφορά μεταξύ μελών δεδομένων και μεθόδων

Τα μέλη λένε το **τι είναι μια κλάση (ορίζουν τα χαρακτηριστικά της γνωρίσματα)**, ενώ οι μέθοδοι λένε **τι κάνει μια κλάση (ορίζουν τη συμπεριφορά της)**. Στο προηγούμενο παράδειγμα του website μπορούμε να εισάγουμε μια μέθοδο που να τυπώνει τα δεδομένα όπως παρακάτω:

```
class website
{
    String name;
    String url;
    String description;

    print()
    {
        System.out.println (name + " at " + url + " is " + description);
    }

}
```

Έξω από τη μέθοδο website καλούμε τη μέθοδο print, ακριβώς όπως αναφερόμασταν στις μεταβλητές μέλους, χρησιμοποιώντας το όνομα του συγκεκριμένου αντικειμένου που θέλουμε να τυπώσουμε και τον τελεστή '.' δηλαδή:

```
website x = new website();
x.name = "Ειδήσεις-Νέα";
x.url = "http://www.in.gr/";
x.description = "Ενημερωτικό site";
x.print();
```

Παρατηρούμε ότι **μέσα στην κλάση του website δεν χρειάζεται να χρησιμοποιούμε x.name ή x.url, name και url είναι αρκετά**. Αυτό ισχύει γιατί η **μέθοδος print πρέπει να καλείται από ένα συγκεκριμένο στιγμιότυπο κλάσης του website**. Η μέθοδος print() έχει

ολοκληρωτικά **ενθυλακωθεί** μέσα στην κλάση του website. **Όλοι οι μέθοδοι στη Java πρέπει να ανήκουν σε μια κλάση**.

1.9 Κατασκευαστής ή Δημιουργός (Constructor)

Η πρώτη μέθοδος που χρειάζονται οι περισσότερες κλάσεις είναι ο κατασκευαστής ή δημιουργός. Ο κατασκευαστής **δημιουργεί ένα νέο στιγμιότυπο** της κλάσης. **Αρχικοποιεί όλες τις μεταβλητές και εκτελεί οποιαδήποτε ενέργεια χρειάζεται για να προετοιμαστεί η κλάση και να χρησιμοποιηθεί**. Στην εντολή websitex = ***new***website(); **η website( )** είναι μια **συνάρτηση κατασκευής**. **Αν δεν υπάρχει τέτοια συνάρτηση η Java δημιουργεί εξορισμού μια, αλλά είναι καλύτερο να έχουμε ορίσει τη δική μας**. Ο κατασκευαστής γράφει μια ***public*** μέθοδο, η οποία έχει το ίδιο όνομα με την κλάση. Έτσι, η δομή του website μας καλείται website(). Εδώ είναι μια απλή website κλάση με μια δομή που αρχικοποιεί όλα τα μέλη με μηδενικά strings.

```
class website
{
   String name;
   String url;
   String description;

   public website()
   {
      name = "";
      url = "";
      description = "";
   }
}
```

Ακόμα καλύτερα πρέπει να δημιουργήσουμε ένα κατασκευαστή που να δέχεται τρία strings σαν ορίσματα και να τα χρησιμοποιεί για να αρχικοποιήσει τις μεταβλητές μέλους ως εξής:

```
class website
{
   String name;
   String url;
```

```
    String description;

    public website(String n, String u, String d)
    {
        name = n;
        url = u;
        description = d;
    }
}
```

Στην περίπτωση αυτή ένα νέο στιγμιότυπο της κλάσης δημιουργείται ως εξής:

websitex = *new*website("Ειδήσεις-Νέα ", " http://www.in.gr /", " Ενημερωτικό site");

x.print(); //καλείται η μέθοδος print() για το στιγμιότυπο αυτό

Τι συμβαίνει όμως όταν θέλουμε να δημιουργήσουμε ένα website του οποίου μερικές φορές ξέρουμε το url, το όνομα και την περιγραφή και μερικές φορές δεν το ξέρουμε; Στην περίπτωση αυτή είναι καλύτερο να δημιουργήσουμε δύο κατασκευαστές όπως φαίνεται ακολούθως:

```
class website
{
    String name;
    String url;
    String description;

    public website(String n, String u, String d)
    {
        name = n;
        url = u;
        description = d;
    }
```

```
    public website()
    {
        name = "";
        url = "";
        description = "";
    }
}
```

Αυτό ονομάζεται **υπερφόρτωση μεθόδων** (methodoverloading). Η υπερφόρτωση είναι ένα χαρακτηριστικό των αντικειμενοστραφών γλωσσών **που επιτρέπουν στο ίδιο όνομα να αναφέρεται σε διαφορετικές μεθόδους, οι οποίες διαφοροποιούνται μεταξύ τουςστον αριθμό και/ή στον τύπου των ορισμάτων τους**. Η υπερφόρτωση μεθόδων εξηγείται αναλυτικά στο επόμενο κεφάλαιο.

Στο προηγούμενο παράδειγμα χρησιμοποιούμε την πρώτη έκδοση της μεθόδου αν έχουμε τρία ορίσματα και την δεύτερη έκδοση αν δεν έχουμε καθόλου ορίσματα. Αν έχουμε ένα, δύο ή τέσσερα ορίσματα τύπου ***String*** ή ορίσματα που δεν είναι ***String***, ο compiler δημιουργεί ένα λάθος γιατί δεν έχει μια μέθοδο που να ταιριάζει με τη μέθοδο που ζητήθηκε.

1.10 Υπερφόρτωση μεθόδων

Η Java δε διακρίνει τις μεθόδους μόνο από το όνομά τους. Χρησιμοποιεί αυτό που ονομάζεται «πλήρως προσδιορισμένο όνομα μεθόδου» (fullyqualifiedmethodname). Το πλήρως προσδιορισμένο όνομα μιας μεθόδου περιλαμβάνει το όνομά της, την κλάση της και τα αθροίσματά της. Δύο μέθοδοι μπορούν να έχουν το ίδιο όνομα αρκεί να είναι διαφορετικά τα πλήρως προσδιορισμένα ονόματά τους. Η δημιουργία **δύο μεθόδων με το ίδιο όνομα αλλά διαφορετικά ορίσματα ονομάζεται υπερφόρτωση μεθόδων (methodoverloading)**.

1.10.1 Ίδια ονόματα μεθόδων σε μια κλάση

Ακόμα και μέσα σε μια κλάση, δύο μέθοδοι μπορούν να έχουν το ίδιο όνομα, εφόσον μπορεί να γίνει διάκριση μεταξύ τους με βάση τα ορίσματά τους. Έτσι τα επόμενα είναι έγκυρα:

```
class BankAccount
(
// το επιτόκιο του λογαριασμού
    double currentinterestrate;

    double rate() //rate (επιτόκιο) επιστρέφει ή καθορίζει το τρέχον επιτόκιο
    {
        return currentinterestrate;
```

```
    }

    double rate(double newrate)
    {
        if (newrate >.0.0 && newrate < 20.0)
            currentinterestrate = newrate;

        return currentinterestrate;
    }

    //τα υπόλοιπα όπως πριν…..
)
```

Στη μέθοδο rate έχει γίνει υπερφόρτωση (overloading) ώστε να πραγματοποιεί δύο σχετικές μεταξύ τους λειτουργίες. Η rate επιστρέφει το τρέχον επιτόκιο ενώ η μέθοδος rate (***double***) καθορίζει το τρέχον επιτόκιο. Στην πράξη είναι εύκολο να τις ξεχωρίσουμε:

```
double fn( BankAccountbamyaccount)
(
   bamyaccount.rate(8.0);//καθορίζει το επιτόκιο
   returnbamyaccount.rate()//εξακρίβωση του επιτοκίου
}
```

Όπως αναφέραμε η δημιουργία δύο μεθόδων με το ίδιο όνομα αλλά διαφορετικά ορίσματα ονομάζεται υπερφόρτωση μεθόδων. **Ωστόσο δεν είναι δυνατόν να διακρίνουμε δύο μεθόδους με βάση τον επιστρεφόμενο τύπο τους**. Και αυτό γιατί συχνά είναι αδύνατον να καθορίσουμε τον επιστρεφόμενο τύπο αποκλειστικά και μόνον από τον τρόπο με τον οποίο χρησιμοποιείται η μέθοδος. Ας υποθέσουμε για παράδειγμα ότι επιτρέπονται τα εξής:

```
class realnumber
(
   //μετέτρεψε τον τρέχοντα αριθμό RealImaginary σε double
   double convert();

   //μετέτρεψε τον τρέχοντα αριθμό RealImaginary σε int
```

```
    int convert();
)
```

//προκαλεί το εξής πρόβλημα:

```
    void fn(realnumberrn)
    (
        rn.convert(); //ποια συνάρτηση από όλες;
    )
```

Επιτρέπεται να αγνοήσουμε την τιμή που επιστέφεται από μια μέθοδο αλλά κάνοντάς το αυτό στερούμε το μεταγλωττιστή από ένα τρόπο με τον οποίο θα μπορούσε να καθορίσει ποια μέθοδο convert προσπαθεί να καλέσει ο καλών. Για να αποφύγουμε την κατάσταση αυτή η Java απλώς αποκλείει τον επιστρεφόμενο τύπο από το πλήρως προσδιορισμένο όνομα. Έτσι ο δεύτερος ορισμός του realnumber.convert προκαλεί τη γένεση ενός σφάλματος μεταγλωττιστή.

1.11 Μέθοδος toString

Η μέθοδος print είναι κοινή σε κάποιες γλώσσες, αλλά τα περισσότερα προγράμματα της Java την χειρίζονται διαφορετικά. Μπορούμε να χρησιμοποιήσουμε την εντολή ***System.out.println*()** για να τυπώσουμε κάποιο αντικείμενο. Όμως η java μας προσφέρει μια μέθοδο με όνομα **toString()** που **μορφοποιεί τα δεδομένα των αντικειμένων** με ένα συγκεντρωτικό τρόπο και επιστρέφει ως αποτέλεσμα ένα ***String***.

Παρακάτω βλέπουμε πως χρησιμοποιείται η μέθοδος toString()στο παράδειγμα του website:

```
    class website
    {
        String name;
        String url;
        String description;

        public website(String n, String u, String d)
        {
            name = n;
            url = u;
```

```
        description = d;
    }

    public website()
    {
        name = "";
        url = "";
        description = "";
    }

    public String toString()
    {
        return (name + " at " + url + " is " + description);
    }
}

public class ClassTest
{
    public static void main(Stringargs[])
    {
        websitex = newwebsite("Ειδήσεις-Νέα ", " http://www.in.gr /", " Ενημερωτικό site");
        System.out.println(x);
    }
}
```

Η εκτύπωση μας στο τελευταίο πρόγραμμα ήταν λίγο εξεζητημένη γιατί χρειαστήκαμε να «σπάσουμε» έναν αριθμό complex στο πραγματικό και στο φανταστικό του μέρος, να τον τυπώσουμε και μετά να τον τοποθετήσουμε ξανά πίσω. Δεν θα ήταν πιο απλό, θα ρωτούσε κάποιος, να μπορούσαμε απλά να γράψουμε: ***System.out.println***(u);

Αποδεικνύεται ότι μπορούμε. Όλα τα αντικείμενα έχουν μια μέθοδο toString που προέρχεται από την Object***class***. Όμως η εξορισμού μέθοδος toString() δεν είναι πολύ χρήσιμη έτσι πρέπει να την υπερβούμε και να την προσαρμόσουμε στους μιγαδικούς αριθμούς. Προσθέτουμε την ακόλουθη μέθοδο στην κλάση Complex:

```
publicStringtoString()
{
    if (v >= 0)
        return (String.valueOf(u) + " + " + String.valueOf(v) + "i");
    else
        return (String.valueOf(u) + " - " + String.valueOf(-v) + "i");
}
```

Πρέπει επίσης να προσθέσουμε την **class**ComplexExamples όπως παρακάτω:

```
class ComplexExamples
{
    publicstatic void main (Stringargs[])
    {
      Complex u, v, z;

      u = new Complex(1,2);
      System.out.println ("u: " + u);
      v = new Complex(3,-4.5);
      System.out.println ("v: " + v);

      //Add u + v;
      z=u.Plus(v);
      System.out.println ("u + v: " + z);
      // Add v + u;
      z=v.Plus(u);
      System.out.println ("v + u: " + z);

      z=u.Minus(v);
      System.out.println ("u - v: " + z);
      z=v.Minus(u);
      System.out.println ("v - u: " + z);
```

```
      z=u.Times(v);
      System.out.println ("u * v: " + z);
      z=v.Times(u);
      System.out.println ("v * u: "+ z);

      z=u.DivideBy(v);
      System.out.println ("u / v: " + z);
      z=v.DivideBy(u);
      System.out.println ("v / u: " + z);
   }
}
```

Μέχρι τώρα οι μέθοδοι μας έκαναν πράξεις με δύο μιγαδικούς αριθμούς. Είναι συνηθισμένο να θέλουμε να πολλαπλασιάσουμε ένα μιγαδικό αριθμό με ένα πραγματικό. Για να προσθέσουμε αυτή τη δυνατότητα στην κλάση μας θα χρειαστεί να προσθέσουμε την ακόλουθη μέθοδο:

```
      public Complex Times (double x)
      {
         return new Complex(u*x, v*x);
      }
```

Εδώ είναι ένα απλό πρόγραμμα για τη νέα μέθοδο:

```
class RealComplex
{
   public static void main (String args[])
   {
     Complex v, z;
     double x = 5.1;

     v = new Complex(3,-4.5);
     System.out.println ("v: " + v);
```

```
        System.out.println ("x: " + x);

        z=v.Times(x);
        System.out.println ("v * x: " + z);
    }
}
```

1.12 ΑφηρημένεςΚλάσεις

Υπάρχουν μερικές κλάσεις για τις οποίες δεν έχει νόημα η δημιουργία αντικειμένων. Αυτές οι κλάσεις θα πρέπει να δηλώνονται ως abstract (αφηρημένες). Οι κλάσειςπου δηλώνονται ως abstract δεν μπορούν να χρησιμοποιηθούν για την δημιουργία αντικειμένων και χρησιμοποιούνται μόνο σαν υπερκλάσειςάλλων κλάσεων. Για να είναι μια κλάση αφηρημένη πρέπει να έχει μια αφηρημένη μέθοδο δηλ μια μέθοδο που δεν υλοποιείται στην κλάση αυτή αλλά σε παραγόμενες κλάσεις

Όταν μία κλάση έχει μία αφηρημένη (abstract) μέθοδο τότε

Δεν δίνει παρά μόνο την δήλωση της μεθόδου, χωρίς να την ορίζει. Η δήλωση αυτή θα πρέπει να περιέχει και την δήλωση abstract όπως φαίνεται στο ακόλουθο τμήμα κώδικα:

publicabstractvoidanAbstractMethod();

Δεν μπορούμε να δημιουργήσουμε αντικείμενα από αυτή την κλάση η οποία θεωρείται και πρέπει να δηλωθεί ως abstract όπως στο ακόλουθο παράδειγμα κώδικα:

public **abstract** class AClass

Όλες οι υποκλάσεις της τάξης αυτής, θα πρέπει να ορίσουν την ***abstract*** μέθοδο, αλλιώς και αυτές με τη σειρά τους θα είναι ***abstract*** και δεν θα μπορούμε να δημιουργήσουμε αντικείμενα

Ακολουθεί ένα παράδειγμα με αφηρημένη κλάση

```
abstract public class Shape
{
   public abstract double area();
   public abstract double circumferance();
   public abstract void printState();
}

public class Rectangle extends Shape
{
   protected double x1,y1,x2,y2;
   protected double w,h;

   public Rectangle (double x1, double y1, double x2, double y2)
   {
      this.x1=x1;
      this.y1=y1;
      this.x2=x2;
      this.y2=y2;
      w=x2-x1;
      h=y2-y1;
   }
```

```
    public double area() {return (w*h);}
    public double circumferance() {return(2*(w+h));}

    public void printState()
    {
        System.out.println ("rectangle state:\n");
        System.out.println ("w="+w+"\n");
        System.out.println ("h="+h+"\n");
    }
}

public class Circle extends Shape
{
    protected double x,y;
    protected double r;

    public Circle (double x, double y, double r)
    {
        this.x=x;
        this.y=y;
        this.r=r;
    }

    public double area() {return (3.14*r*r);}
    public double circumferance() {return(2*3.14*r);}

    public void printState()
    {
        System.out.println ("circle state:\n");
        System.out.println ("r="+r+"\n");
    }
}

public class Trans
{
    int type;
    Object obj;

    public Trans(Object obj, int type)
    {
        this.obj=obj;
        this.type=type;
    }
}

import java.util.Stack;

public class ergA
{
  public static void main(String [] args)
```

```
  {
    Stack s= new Stack();
    int i = 0;

    Circle c1= new Circle(1.0,1.0,2.0);
    Trans instanc0 = new Trans(c1,0);
    s.push((Circle)c1);

    Rectangle r1= new Rectangle(1.0,1.0,3.0,3.0);
    Trans instanc1 = new Trans(r1,1);
    s.push((Rectangle)r1);

    Circle c2= new Circle(2.0,4.0,1.0);
    Trans instanc2 = new Trans(c2,1);
    s.push((Circle)c1);

    Rectangle r2= new Rectangle(1.0,0.0,4.0,5.0);
    Trans instanc3 = new Trans(r2,1);
    s.push((Rectangle)r2);

    Object obj;

    for (i=1;i<=4;i++)
    {
          obj=s.pop();
          if (obj instanceof Circle)
              System.out.println ("Pop a circle");
          else
              if (obj instanceof Rectangle)
                  System.out.println ("Pop a rectangle");
    }
  }
}
```

1.13 Διεπαφές

Μία διεπαφή (interface) είναι ένα σύνολο δηλώσεων μεθόδων. Αυτές οι μέθοδοι δεν υλοποιούνται στην διεπαφή, απλά δηλώνονται. Στη συνέχεια η διεπαφή μπορεί να υλοποιηθεί από κάποιες τάξεις. Μία τάξη που υλοποιεί μία διεπαφή θα πρέπει να υλοποιήσει όλες ή κάποιες από τις μεθόδους που δηλώνονται στη διεπαφή.

Το μόνο που κληρονομεί μία τάξη από την διεπαφή που υλοποιεί είναι ο τύπος της. Έτσι αποφεύγονται πολλά από τα προβλήματα που παρουσιάζονται με την κληρονομικότητα υλοποίησης (implementationinheritance) και κυρίως το σημασιολογικό πρόβλημα της εύθραυστης τάξης βάσης (semanticfragilebaseclassproblem) που συζητήσαμε σύντομα στο τέλος της προηγούμενης ενότητας.

Συνοψίζοντας ένα interfaceμοιάζει με μία abstract κλάση αλλά έχει τις εξής διαφορές:

1)τα interfaces ορίζονται έξω απο τη ιεραρχία της κληρονομικότητας. Δηλαδή κάθε κλάση μπορεί να υλοποιήσει ένα ***interface*** χωρίς να κληρονομείται απο αυτό.

2)μια κλάση μπορεί να υλοποιήσει διάφορα interfaces (περισσότερα απο ένα),άρα τα ***interface*** έχουν την δυνατότητα να υποστηρίξουν την πολλαπλή κληρονομικότητα.

3)η κληρονομικότητα απο ένα ***interface*** γίνεται με την λέξη ***implements*** αντί για την λέξη ***extends***.

4)η κλάση που υλοποιεί ένα ***interface*** πρέπει να υλοποιεί ΥΠΟΧΡΕΩΤΙΚΑ ΟΛΕΣ τις μεθόδους του interface, δεν πρέπει ΟΜΩΣ ΝΑ ΧΡΗΣΙΜΟΠΟΙΕΙ ΥΠΟΧΡΕΩΤΙΚΑ ΟΛΑ ΤΑ ΔΕΔΟΜΕΝΑ-ΜΕΛΗ ΤΟΥ INTERFACE (π.χ. τις σταθερές του)

5)κάθε κλάση που υλοποιεί το INTERFACE ΠΡΕΠΕΙ ΜΠΡΟΣΤΑ ΑΠΟ ΟΛΕΣ ΤΙΣ ΜΕΘΟΔΟΥΣ ΤΟΥ INTERFACE ΠΟΥ ΥΛΟΠΟΙΕΙ ΝΑ ΕΧΕΙ ΥΠΟΧΡΕΩΤΙΚΑ ΤΗ ΛΕΞΗ PUBLIC

Ένα παράδειγμα χρήσης διεπαφής είναι το ακόλουθο:

```
interface methods //δήλωσηinterface μεόνομαmethods
{ //το συγκεκριμένο interfaceπεριλαμβάνει μόνο αφηρημένους μεθόδους
    void setcolor(String color);
    void setcenter(int x, int y);
    void setside(double r);
    void print();
    double area();
}

interface constants//δήλωσηinterface μεόνομαconstants
{//το συγκεκριμένο interfaceπεριλαμβάνει μόνο μια σταθερά
    finaldouble p=3.14;
}

class circle implements methods, constants //κλάσηπουκληρονομεί 2 interfaces
{
    privateint x,y;
    privatedouble radius;
```

```
    privateString color;

    publicvoid setcolor(String color)
    {
        this.color= color;
    }

    publicvoid setcenter(int x, int y)
    {
        this.x= x;
        this.y= y;
    }

    publicvoid setside(double x)
    {
        radius=x;
    }

    publicvoid print()
    {
        System.out.println ("ο κύκλος έχει ακτίνα "+radius+"κέντρο "+x+","+y+" χρώμα "+color);
    }

    publicdouble area()
    {
        return p*radius*radius;
    }

}

publicclass interface1
{
    publicstaticvoidmain(String[] args)
    {
        circle c1= new circle();

        c1.setcenter(1,1);
        c1.setside(2.0);
        c1.setcolor("red");
        c1.print();

        System.out.println ("εμβαδόν="+c1.area());
    }
}
```

δεν μπορούμε να δημιουργήσουμε στιγμιότυπα σε ***interface*** όπως και σε abstract κλάση διότι οι μέθοδοι τους δεν είναι ολοκληρωμένοι

1.14 Κληρονομικότητα

Η έννοια της κλάσης που έχουμε δει μέχρι τώρα φαίνεται να είναι απλώς ένα οργανωτικό εργαλείο. Αυτό που δίνει στην κλάση την εκφραστική της δύναμη είναι η έννοια της κληρονομικότητας (inheritance). Μια κλάση κληρονομεί μέλη δεδομένων και μεθόδους από κάποια άλλη κλάση με τη λέξη κλειδί ***extends***. Ας δούμε ένα παράδειγμα κληρονομικότητας προκειμένου αυτή η έννοια να γίνει κατανοητή.

```
class BankAccount
{
   private double balance;

   public double balance()
   {
      return balance;
   }
};

class CheckingAccount extends BankAccount
{
   public void WriteCheck(double amount)
   {
   }
};
```

Η κλάση CheckingAccount (λογαριασμός όψεως) είναι μια νέα κλάση που **κληρονομεί** από την κλάση BankAccount. Μπορούμε επίσης να πούμε ότι η CheckingAccount **επεκτείνει** (***extends***)την κλάση BankAccount. Μπορούμε επίσης να πούμε ότι η κλάση CheckingAccount είναι μια **υπο-κλάση** (subclass) της BankAccount ή ότι η BankAccount είναι μια **υπερ-κλάση** (super ***class***) ή **βασική κλάση** (base ***class***) της CheckingAccount.). Με αυτή τη δήλωση ένας λογαριασμός όψεως CheckingAccount) είναι ένας τραπεζικός λογαριασμός(BankAccount). Έτσι ο κώδικας που ακολουθεί είναι επιτρεπτός:

```
class MyClass
{
   public static void SomeFunc()
   {
        CheckingAccount ca= new CheckingAccount();

        ca.deposit(100000); //κατάθεσηστολογαριασμόόψεως
        System.out.println ("Υπόλοιπολογαριασμούόψεως=" +ca.balance());
        ca.deposit(10000); //ανάληψη από το λογαριασμό όψεως
   }
}
```

Η συνάρτηση balance (υπόλοιπο) δε δηλώνεται μέσα στην κλάση CheckingAccount. Ωστόσο η CheckingAccount κληρονομεί τα μέλη της BankAccount μεταξύ των οποίων και την balance(). Όταν καλέσουμε την ca.balance() καλείται η συνάρτηση balance μέσα στην BankAccount. Αυτό ταιριάζει με την αντίληψή μας για την κληρονομικότητα αφού ένας λογαριασμός όψεως είναι τραπεζικός λογαριασμός έχει όλα τα δικαιώματα και τα προνόμια ενός τραπεζικού λογαριασμού.

1.15 Υπέρβαση μεθόδων

Μια υποκλάση μπορεί να **υπερβεί** (override) μια μέθοδο της υπερκλάσης. **Υπέρβαση μεθόδου σημαίνει να αντικαταστήσει μια μέθοδο που έχει κληρονομηθεί σε αυτή με μια δική της μέθοδο**. Για παράδειγμα ας υποθέσουμε ότι ο λογαριασμός όψεως τοκίζεται με διαφορετικό τρόπο από τα άλλα είδη λογαριασμών και συγκεκριμένα απαιτείται ένα ελάχιστο υπόλοιπο 5000 για να αρχίσει να τοκίζεται ο λογαριασμός όψεως. Μπορούμε να υλοποιήσουμε τη μέθοδο αυτή υπερβαίνοντας τη μέθοδο interest της κλάσης CheckingAccount ως εξής:

```
class CheckingAccount extends BankAccount
{
   public void Interest() //Τόκος- πρόσθεσετομηνιαίοτόκο
   {
      //οι λογαριασμοί όψεως πρέπει να περιέχουν ένα ελάχιστο ποσόν 5000 για να τοκίζονται
      if (balance()>=5000)
      {
      ……
      }
}
```

Τώρα η κλάση CheckingAccount κληρονομεί όλες τις μεθόδους της BankAccount εκτός από τη μέθοδο interest για την οποία διαθέτει τη δική της λύση. Αυτό μας βάζει ένα ενδιαφέρον πρόβλημα. Εξετάζουμε και πάλι τη γενική συνάρτησή μας SomeFunc(BankAccount). Ας υποθέσουμε ότι η συνάρτηση αυτή ονομάζεται Interest όπως στα επόμενα:

1.16 Πολυμορφισμός

Η έννοια του πολυμορφισμού βασίζεται στον τρόπο με τον οποίο οι άνθρωποι χτίζουν και χρησιμοποιούν αντικείμενα στον πραγματικό κόσμο. Στον πραγματικό κόσμο κάνουμε υποθέσεις βασισμένοι σε γεγονότα που γνωρίζουμε ότι είναι αληθινά. Κάποια γεγονότα ή μέθοδοι εφαρμόζονται σε όλα τα αντικείμενα κάποιου συγκεκριμένου τύπου και παρόλο που οι διεργασίες με τις οποίες τα γεγονότα αυτά εφαρμόζονται είναι πολλές και πολύπλοκες δεν ασχολούμαστε αναγκαστικά με τις διαφορές. Γνωρίζουμε αυτά που χρειάζεται να γνωρίζουμε.

Ο πολυμορφισμός αυξάνει σημαντικά το επίπεδο της αφαιρετικότητας σε ένα πρόγραμμα. Στο πρόγραμμά μας μπορούμε να αναφερθούμε σε τραπεζικούς λογαριασμούς χωρίς να ανησυχήσουμε που περιλαμβάνονται όλα τα διαφορετικά είδη λογαριασμών. Απλώς αναφερόμαστε στην κλάση BankAccount.

Εκτός του ότι γίνεται ευκολότερη η ζωή των προγραμματιστών αυτή η αύξηση της αφαιρετικότητας έχει πρόσθετο αποτέλεσμα τη βελτίωση της συντήρησης των εφαρμογών. Οι πληροφορίες για τη σχέση ενός λογαριασμού όψεως με ένα τραπεζικό λογαριασμό απομονώνονται σε ένα μέρος: στο πακέτο των τραπεζικών λογαριασμών. Για την ακρίβεια αν οι κλάσεις σχεδιαστούν σωστά το υπόλοιπο πρόγραμμα δε γνωρίζει ούτε την ύπαρξη του αντικειμένου που ονομάζεται λογαριασμός όψεως ούτε ότι αυτός έχει διαφορετικούς κανόνες από ένα τραπεζικό λογαριασμό.

Τέλος, ο πολυμορφισμός αυξάνει τη δυνατότητα μας να ξαναχρησιμοποιήσουμε τις κλάσεις. Με λίγα λόγια ο πολυμορφισμός κάνει την κληρονομικότητα να δουλεύει.

1.17 Εξαιρέσεις

Η javaέχει τη δυνατότητα να αντιμετωπίζει σφάλματα που συνέβησαν κατά την εκτέλεση του κώδικα. Συγκεκριμένα μπορεί να εντοπίζει το είδος του λάθους που συνέβη και να εκτυπώνει ένα αντίστοιχο διαγνωστικό μήνυμα πληροφορώντας το χρήστη για το τι ακριβώς συνέβη. Αυτό βέβαια δεν σημαίνει ότι κάνει διόρθωση λαθών αλλά βοηθά το χρήστη να αντιληφθεί τα λάθη που έχει κάνει έχει ώστε να μπορέσει στη συνέχεια να τα διορθώσει μόνος του.

Ο τρόπος αντιμετώπισης-σύλληψης μιας εξαίρεσης γίνεται με τις ακόλουθες εντολές:

```
try
{
        εντολή 1;
        εντολή 2;
        …….
        εντολή ν;
}
catch (Τύπος Σφάλματος μεταβλητή σφάλματος)
{
        εντολή 1;
        εντολή 2;
        …….
        εντολή ν;
}
```

εναλλακτικά μπορούμε να γράψουμε:

```
try
{
        εντολή 1;
        εντολή 2;
        …….
        εντολή ν;
}
catch (Τύπος Σφάλματος μεταβλητή σφάλματος)
{
        εντολή 1;
```

εντολή 2;

…….

εντολή ν;

}

finally

{

εντολή 1;

εντολή 2;

…….

εντολή ν;

}

Μέσα στο tryβάζουμε τον κώδικα που ενδέχεται κατά την εκτέλεση του να προκαλέσει σφάλμα (εξαίρεση) και στο catchγράφουμε τις εντολές για την αντιμετώπιση του σφάλματος. Στο catchγράφουμε τον τύπο του σφάλματος και μπορούμε σε ένα tryνα αντιστοιχήσουμε περισσότερα από ένα catch, ένα για κάθε τύπου σφάλματος. Αν ο κώδικας στο tryδεν παρουσιάσει κανένα σφάλμα τότε προφανώς ο κώδικασμέσα στο catchδεν θα εκτελεστεί.Επίσης μπορούμε στο τέλος όλων των catchνα προσθέσουμε και μια εντολή finallyη οποία περιλαμβάνει ένα κώδικα που θα εκτελεστεί σε κάθε περίπτωση είτε συμβεί σφάλμα είτε όχι.

Παράδειγμα

Ο ακόλουθος κώδικας όταν εκτελείται τυπώνει το μήνυμαHello και περιμένει να λάβει ένα όρισμα από τη γραμμή εντολών

classHello

{

publicstaticvoidmain (***String*** [] ***args***)

{

System.out.print("Hello ");

System.out.println(***args***[0]);

}

}

Αν τρέξουμε το πρόγραμμα χωρίς να δώσουμε όρισμα από τη γραμμή εντολών τότε το σύστημα θα δημιουργούσε την ακόλουθη εξαίρεση.

Exception in thread "***main***" java.lang.ArrayIndexOutOfBounds exception at Hello.mainC:\javahtml\Hello.java:7)

Το μήνυμα αυτό φαίνεται ιδιαίτερα δύσκολο ώστε να γίνει αντιληπτό από ένα προγραμματιστή ειδικά άπειρο. Αλλά και έμπειρος να είναι ο προγραμματιστής όταν σε ένα μεγάλο πρόγραμμα εμφανίζονται πολλά τέτοια μηνύματα και πάλι είναι δύσκολο να τα αντιληφθεί και κυρίως να καταλάβει τι ακριβώς συνέβη

Ακολουθεί ένα παράδειγμα χειρισμού εξαιρέσεων σε Java που εφαρμόζεται στο πρόγραμμα HelloWorld:

```
class ExceptionalHello
{
   public static void main (String [] args)
   {
      try
      {
         System.out.println ("Hello " + args[0]);
      }
      catch (Exception e)
      {
         System.out.println ("Missing command line argument");
      }
   }
}
```

Τώρα αν εκτελέσουμε τον ίδιο κώδικα και δεν δώσουμε και πάλι όρισμα από τη γραμμή εντολών, τότε το σύστημα θα εμφανίσει το μήνυμα:

Missingcommandlineargument

Κάποιες εξαιρέσεις είναι δυνατόν να τις αντιληφθούμε και να τις χειριστούμε, ενώ κάποιες άλλες είναι τόσο δύσκολες που το σύστημα εκτέλεσης παραιτείται. Ο compiler θα «χτυπήσει» αν γράφουμε κώδικα και δεν συμπεριλάβουμε τις όχι και τόσο επικίνδυνες εξαιρέσεις, αλλά εμείς πρέπει να προσέχουμε για τις επικίνδυνες (όπως το ArrayIndexOutOfBoundsExceptions). Πολλές φορές το να αντιληφθούμε την εξαίρεση δεν αρκεί γιατί δεν μπορούμε να την αντιμετωπίσουμε. Οι «κακές» εξαιρέσεις σταματούν το πρόγραμμα. Πολλές φορές το θέλουμε κι εμείς οι ίδιοι. Σε αυτή την περίπτωση καλείται η **System.exit(*int*) μέθοδος και το εκτελούμενο πρόγραμμα κρεμάει**. Άλλες φορές μπορούμε να βγούμε από ένα βρόγχο και να συνεχίσουμε με το υπόλοιπο πρόγραμμα. Αυτό είναι συνηθισμένο όταν μια εξαίρεση είναι αναμενόμενη ή όταν δεν επηρεάζει αρνητικά τη λογική του προγράμματος. Μπορούμε να τυπώνουμε ένα μήνυμα λάθους ή όχι. Αν γράφουμε ένα χειριστή εξαιρέσεων και δεν περιμένουμε να τον καλούμε τότε γράφουμε την εντολή:

```
System.out.println ("Error: " + e);
```

Με αυτό τον τρόπο αν κάτι πάει λάθος, τουλάχιστον θα γνωρίζουμε που συνέβη το λάθος.

1.18 Αρχεία I/O και streams

Πολλές φορές γράφουμε δεδομένα σε ένα αρχείο αντί για την οθόνη του υπολογιστή. Κάτω από UNIX ή DOS μπορούμε να το κάνουμε αυτό χρησιμοποιώντας τους τελεστές <και >. Κάποιες φορές χρειαζόμαστε μια άλλη προσέγγιση που να γράφει συγκεκριμένα δεδομένα σε ένα αρχείο ενώ όλα τα άλλα στην οθόνη ή χρειαζόμαστε πρόσβαση σε διάφορα αρχεία ταυτοχρόνως ή ίσως θέλουμε να διαβάσουμε τα δεδομένα ενός αρχείου που έχουν μια συγκεκριμένη μορφοποίηση. **Στη Java όλες αυτές οι μέθοδοι συμβαίνουν σαν streams**.

1.19 Άμεση επικοινωνία με το χρήστη

Θα ξεκινήσουμε με ένα πολύ απλό πρόγραμμα το οποίο ζητά τα ονόματα των χρηστών και έπειτα τυπώνει ένα προσωπικό χαιρετισμό.

```
import java.io.*;


class PersonalHello
{
   public static void main (String [] args)
   {
     byte name[] = new byte[100];
     int nr_read = 0;


     System.out.println ("What is your name?");
     try
     {
        nr_read = System.in.read(name);
        System.out.print("Hello ");
        System.out.write(name,0,nr_read);
     }
     catch (IOException e)
     {
        System.out.print("I'm Sorry. I didn't catch your name.");
     }
   }
}
```

Το *import*.java.io.* είναι το αντίστοιχο #include<stdio. h> της C. Τα περισσότερα από αυτά που διαβάζουμε και γράφουμε στη Java είναι σε bytes. (Αργότερα θα δούμε ότι γίνεται να διαβάζουμε αρχεία κειμένου σε strings). Εδώ ξεκινήσαμε με ένα πίνακα από bytes που θα κρατάει το όνομα του χρήστη. Πρώτα τυπώνουμε ένα ερώτημα ζητώντας το όνομα του χρήστη. Έπειτα διαβάζουμε το όνομα χρησιμοποιώντας τη μέθοδο System.in.read(). Αυτή η μέθοδος παίρνει ένα πίνακα byte σαν όρισμα και τοποθετεί αυτό που πληκτρολογεί ο χρήστης. Έπειτα όπως και πριν τυπώνουμε «Hello». Τέλος τυπώνουμε το όνομα του χρήστη. Το πρόγραμμα στην πραγματικότητα δεν βλέπει τι πληκτρολογεί ο χρήστης μέχρι αυτός να πατήσει το enter. Έτσι δίνεται η ευκαιρία στον χρήστη να διορθώσει τυχόν λάθη. Όταν όμως πατήσει το πλήκτρο enter, ό,τι υπάρχει στη γραμμή τοποθετείται στον πίνακα. Τι γίνεται αν ο χρήστης πληκτρολογήσει περισσότερους από 100 χαρακτήρες προτού πατήσει το enter; Σε πολλές προγραμματιστικές γλώσσες αυτό θα οδηγούσε στο σπάσιμο του προγράμματος. Η Java είναι πιο ασφαλής. Η μέθοδος System.in.read() ελέγχει το μήκος του πίνακα χρησιμοποιώντας την ιδιότητα name.length.

1.20 Διαβάζοντας αριθμούς

Συχνά τα αλφαριθμητικά δεν αρκούν. Πολλές φορές ζητάμε από το χρήστη έναν αριθμό σαν είσοδο. Όλες **οι είσοδοι του χρήστη εισάγονται σαν αλφαριθμητικά**, τα οποία θέλουμε να τα μετατρέψουμε σε αριθμούς. Γράφουμε τη μέθοδο getNextInteger() που δέχεται ένα ακέραιο από το χρήστη.

```
static int getNextInteger()
{
   String line;

   DataInputStream in = newDataInputStream(System.in);
   try
   {
      line = in.readLine();
      int i = Integer.valueOf(line).intValue();
      return i;
   }
   catch (Exception e)
   {
      return -1;
   }
}
```

1.21 Διαβάζοντας μορφοποιημένα δεδομένα

Είναι συχνή η περίπτωση να ζητάμε όχι ένα αριθμό αλλά πολλούς. Άλλες φορές θέλουμε να διαβάσουμε κείμενο και αριθμούς στην ίδια γραμμή. Γιαυτό το σκοπό η Java παρέχει την κλάση StreamTokenizer.

1.22 Γράφοντας ένα αρχείο κειμένου

Κάποιες φορές θέλουμε να σώσουμε μία έξοδο σέρνοντάς την (scrolling) απλώς στην οθόνη. Για να το κάνουμε αυτό πρέπει να μάθουμε πώς να γράφουμε δεδομένα σε ένα αρχείο. Αντί να δημιουργήσουμε ένα καινούριο πρόγραμμα θα τροποποιήσουμε το Fahrenheitto-Celsius.

```
import java.io.*;


class FahrToCelsius
{
   public static void main (String [] args)
   {
      double fahr, celsius;
      double lower, upper, step;


      lower = 0.0; //κάτω όριο θερμοκρασίας
      upper = 300.0; //πάνω όριο θερμοκρασίας
      step = 20.0; // step size
      fahr = lower;
      try
      {
         FileOutputStream fout = newFileOutputStream("test.out");
         PrintStreammyOutput = newPrintStream(fout);


         while (fahr <= upper)
         {
               celsius = 5.0 * (fahr-32.0) / 9.0;
               myOutput.println(fahr + " " + celsius);
               fahr = fahr + step;
         }
```

```
        }
        catch (IOException e)
        {
            System.out.println ("Error: " + e);
            System.exit(1);
        }
    }
}
```

Υπάρχουν 3 απαραίτητες συνθήκες για να γράψουμε μορφοποιημένη έξοδο σε ένα αρχείο:

α)**ανοίγουμε ένα FileOutputStream** χρησιμοποιώντας την ακόλουθη εντολή:

```
FileOutputStream fout = new FileOutputStream("test.out");
```

Αυτή η γραμμή **αρχικοποιεί το FileOutputStream** με το όνομα του αρχείου που θέλουμε να γράψουμε μέσα.

β)**μετατρέπουμε το FileOutputStream σε PrintStream** χρησιμοποιώντας την ακόλουθη εντολή:

```
PrintStream myOutput = new PrintStream(fout);
```

γ)αντί να χρησιμοποιήσουμε την εντολή ***System.out.println***() χρησιμοποιούμε την εντολή:

myOutput.***println***().***System.out*** και myOutput.

Είναι διαφορετικά παραδείγματα της κλάσης PrintStream. Για να τυπώσουμε σε ένα διαφορετικό PrintStream κρατάμε ίδια τη σύνταξη αλλά αλλάζουμε το όνομα του PrintStream.

1.23 Διαβάζοντας ένα αρχείο κειμένου

Τώρα που ξέρουμε πώς να γράψουμε ένα αρχείο κειμένου ας προσπαθήσουμε να διαβάσουμε ένα. Ο παρακάτω κώδικας εφαρμόζει το Unix στη Java. Συγκεκριμένα δέχεται μια σειρά από ονόματα αρχείων στη γραμμή εντολών και μετά τυπώνει αυτά τα ονόματα στη συγκεκριμένη έξοδο με τη σειρά που καταγράφηκαν.

```
import java.io.*;
class cat
{
```

```
public static void main (String [] args)
{
  String thisLine;

  for (int i=0; i<args.length; i++)
  {
     try
     {
        FileInputStream fin = new FileInputStream(args[i]);
        try
        {
           DataInputStream myInput = new DataInputStream(fin);
           try
           {
              while ((thisLine = myInput.readLine()) != null)
              {
                 System.out.println (thisLine);
              }
           }
           catch (Exception e)
           {
              System.out.println ("Error: " + e);
           }
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e);
        }
     }
     catch (Exception e)
     {
        System.out.println ("failed to open file " + args[i]);
```

```
            System.out.println ("Error: " + e);
        }
    }
  }
}
```

1.24 Applets

Τα applets (μικροεφαρμογές) είναι προγράμματα Java που μπορούν να «φορτωθούν» σε ιστοσελίδες και να λειτουργήσουν στο Internet. Το περιβάλλον εκτέλεσης (π.χ. ο browser) φροντίζει για την εκτέλεση της εφαρμογής.

Παράδειγμα χρήσης Applet

Ο λόγος για τον οποίο όλοι θεωρούν τη Java ως μια από τις καλύτερες γλώσσες προγραμματισμού είναι ότι επιτρέπει να γράφουμε αλληλεπιδραστικές εφαρμογές στο δίκτυο. Το HelloWorld δεν είναι ένα αλληλεπιδραστικό πρόγραμμα αλλά ας δούμε την παρακάτω έκδοση:

```
import java.applet.Applet;

import java.awt.Graphics;


publicclass HelloWorldApplet extendsApplet
{
   publicvoidpaint(Graphics g)
   {
      g.drawString("Helloworld!", 50, 25);
   }
}
```

Αυτή η έκδοση είναι λίγο πιο πολύπλοκη από την προηγούμενη αλλά θέλει κι άλλη προσπάθεια για να τρέξει καλύτερα. Πρώτα πληκτρολογούμε τον κώδικα και αποθηκεύουμε το σ' ένα αρχείο που να λέγεται HelloWorldApplet.java στο javahtml. Μεταγλωττίζουμε το πρόγραμμα. Αν όλα πάνε καλά θα δημιουργηθεί ένα αρχείο με το όνομα HelloWorldApplet.***class***. Αυτό το αρχείο πρέπει να βρίσκεται στο directory των κλάσεων. Τώρα πρέπει να δημιουργήσουμε ένα αρχείο ***HTML*** που θα συμπεριλαμβάνει την εφαρμογή σας. Ακολουθεί το απλό αρχείο ***HTML***.

```
<HTML>
<HEAD>
<TITLE> Hello World </TITLE>
```

</***HEAD***>

<***BODY***>

This is the applet:<P>

<***APPLET*** codebase="classes" code="HelloWorldApplet.***class***" ***width***=200 ***height***=200 ></***APPLET***>

</***BODY***>

</***HTML***>

Αποθηκεύουμε το αρχείο αυτό με ένα όνομα π.χ. «HelloWorldApplet.html». Μετά φορτώνουμε το αρχείο ***HTML*** σε ένα Javabrowser όπως InternetExplorer και θα δούμε το ακόλουθο μήνυμα:

This is the applet:

Hello World!

Το Hello World Applet πρόσθεσε κάποια πράγματα στο Hello World Application. Κινούμενοι από πάνω προς τα κάτω, το πρώτο πράγμα που προσέχουμε είναι οι δύο γραμμές.

import java.applet.Applet;

import java.awt.Graphics;

Η δήλωση ***import*** στη Java είναι παρόμοια με τη δήλωση #include στην C ή στην C++. Εισάγει τις κλάσεις που περιέχονται σε ένα πακέτο κάπου αλλού. Ένα πακέτο είναι ένα σύνολο από σχετιζόμενες κλάσεις. Σε αυτή την περίπτωση ζητάμε πρόσβαση στις κλάσεις που συμπεριλαμβάνονται στο java.applet.Applet και ***java.awt***.Graphics.

Η επόμενη διαφορά που παρατηρούμε είναι ο ορισμός της κλάσης:

publicclassHelloWorldAppletextendsApplet

Η λέξη κλειδί ***extends*** υπονοεί ότι αυτή η κλάση είναι μια υποκλάση της κλάσης Applet ή αλλιώς η Applet είναι η υπερκλάση του HelloWorldApplet. Η κλάση Applet προσδιορίζεται στο πακέτο java.applet.Applet. Εφόσον το HelloWorldApplet είναι υποκλάση του Applet τότε το HelloWorldApplet κληρονομεί όλη τη λειτουργικότητα του αρχικού Applet.

Η επόμενη διαφορά είναι λιγότερο ορατή. Δεν υπάρχει κύρια μέθοδος. Τα applets δεν τις χρειάζονται. Η κύρια μέθοδος είναι στον browser ή στον AppletViewer όχι στο ίδιο το applet. Τα applets προσφέρουν επιπλέον λειτουργικότητα αλλά δεν μπορούν να τρέξουν χωρίς να τις φιλοξενήσει ένα κύριο πρόγραμμα.

Αντί να αρχίσουν από ένα συγκεκριμένο σημείο του κώδικα τα applets οδηγούνται από γεγονότα. Ένα applet περιμένει για ένα γεγονός όπως το πάτημα ενός πλήκτρου ή το κλικ του

ποντικιού και έπειτα εκτελεί το κατάλληλο eventhandler. Εφόσον αυτό είναι το πρώτο μας πρόγραμμα είχαμε μόνο ένα eventhandler, το paint.

Η μέθοδος paint χρησιμοποιεί ένα αντικείμενο Graphics που επιλέγουμε να το λέμε g. Η κλάση Graphics ορίζεται στο πακέτο ***java.awt***.Graphics. Μέσα στη μέθοδο paint καλούμε την μέθοδο **drawString** για να γράψει το αλφαριθμητικό «HelloWorld» στις συντεταγμένες (50,25). Αυτό είναι 50 pixels αριστερά και 25 pixels κάτω από την πάνω αριστερή γωνία της εφαρμογής.

Βιβλιογραφία

Βιβλία

Εισαγωγή στη C++
Συγγραφέας: Γιάννης Τσιομπίκας, Κωνσταντίνος Μαργαρίτης.. Συντάκτης: eBooks4Greeks

Η γλώσσα C σε βάθος
Εκδόσεις: Κλειδάριθμος
Συγγραφέας: Νίκος Μ. Χατζηγιαννάκης

Η γλώσσα C++ σε βάθος
Εκδόσεις: Κλειδάριθμος
Συγγραφέας: Νίκος Μ. Χατζηγιαννάκης

Η γλώσσα προγραμματισμού C
Εκδόσεις: Κλειδάριθμος
Συγγραφέας: Brian W. Kernighan

Java προγραμματισμός
Εκδόσεις: Γκιούρδας Μ. (2010)
Συγγραφέας: Paul J. Deitel

Ασκήσεις - προγράμματα σε C++
Εκδόσεις: Γκιούρδας Μ.
Συγγραφέας: Harvey M. Deitel

Μάθετε τη Java 24 ώρες
Συγγραφέας: ΚΑΝΤΕΝΧΕΝΤ ΡΟΤΖΕΡΣ
Εκδοτικός οίκος: ΓΚΙΟΥΡΔΑΣ Μ.

On line Tutorials

C++ Tutorial, C++ Made Easy: Learning to Program in C++

C Tutorial - C Made Easy

More Advanced C and C++ Language Feature Tutorials

Ηλεκτρονικό εγχειρίδιο της JAVA
Συντάκτης: eBooks4Greeks, Συγγραφέας: Παπαδοπούλου Μαρία