ΤΕΙ Λάρισας

Τμήμα Τεχνολογίας Πληροφορικής & Τηλεπικοινωνιών

Προγραμματισμός III: C++

Σημείωσεις Εργαστηρίου

Μαρία Τσιακμάκη

2009-10

Περίληψη

Σε αυτό το εργαστήρι θα ασχοληθούμε με την γλώσσα προγραμματισμού C++.

Η C++ είναι μια γενικής χρήσης γλώσσα προγραμματισμού, με ιδιαίτερη έμφαση στον προγραμματισμό συστήματος (systems language). Αναπτύχθηκε από τον Bjarne Stroustrup το 1979 στα εργαστήρια της Bell (Bell Laboratories) σαν μια προέκταση της C. Υποστηρίζει

- τον διαδικαστικό προγραμματισμό - procedural programming
- την αφαίρεση δεδομένων - data abstraction
- τον αντικειμενοστραφή προγραμματισμό -object-oriented programming, και
- τον γενικευμένο προγραμματισμό - generic programming

Έτσι με τη C++ μπορεί κανείς να

- σχεδιάσει ένα πρόβλημα γύρω από αντικείμενα (objects). Κάθε αντικείμενο μπορεί να έχει ιδιότητες και συμπεριφορές
- οργανώσει τα αντικείμενά του συνθέτοντας κλάσεις (class)
- αποκρύψει ιδιότητες και συμπεριφορές μιας κλάσης από άλλες κλάσεις και συναρτήσεις (ενθυλάκωση - encapsulation)
- αρχικοποιήσει και επαναχρησιμοποιήσει τις κλάσεις όσες φορές χρειάζεται (reusability)
- οργανώσει μια κλάση (βασική κλάση) σε υπο-κλάσεις, ορίζοντας μια ιεραρχημένη δομή κλάσεων σύμφωνα με την οποία κάθε υπό-κλάση μπορεί να κληρονομεί ιδιότητες και συμπεριφορές της βασικής κλάσης (κληρονομικότητα - inheritance)
- χρησιμοποιεί τελεστές και συναρτήσεις με βάση των τύπο των ορισμάτων τους (πολυμορφισμός - polymorphism)
- δώσει την ικανότητα σε υπάρχοντες τελεστές να χειρίζονται νέους τύπους δεδομένων (υπερφόρτωση τελεστών - operator overloading),
- ορίσει κλάσεις που να χειρίζονται διάφορους τύπους δεδομένων (πρότυπα - templates)
- χρησιμοποιήσει κλάσεις, πρότυπα και συναρτήσεις από την καθιερωμένη της βιβλιοθήκη (Standard Template Library - STL)

Στόχος αυτού του εργαστηρίου είναι να σας εισάγει στα παραπάνω στοιχεία μέσα από παραδείγματα και ασκήσεις.

Σημειώστε πως οι σημειώσεις αυτές δεν καλύπτουν το σύνολο του μαθήματος. Για περισσότερα, δείτε τις βιβλιογραφικές αναφορές [2] [3] [5].

Περιεχόμενα

1 Βασικές Έννοιες **4**

2 Δομές και Συναρτήσεις **20**

3 Κλάσεις **38**

4 Υπερφόρτωση τελεστών και φίλες συναρτήσεις **64**

5 Παράγωγες Κλάσεις **78**

6 Συνδυαστικές ασκήσεις **95**

Γλωσσάρι **97**

Βιβλιογραφία **97**

Παράρτημα **98**

A Καθιερωμένοι Χειριστές **99**

Λίστα Προγραμμάτων

1.1 Hello World! 4
1.2 Μετατροπέας δολαρίων σε ευρώ 6
1.3 Αυτόματη Μετατροπή Τύπων 10
1.4 Εκτύπωση βαθμολογίας 11
1.5 For βρόχος 13
1.6 While βρόχος 14
1.7 if...else 15
1.8 Switch...case 16
2.1 Ένα απλό παράδειγμα με δομές 20
2.2 Ένα παράδειγμα με δομές 22
2.3 Ένα παράδειγμα με μαθηματικές συναρτήσεις 25
2.4 Ένα παράδειγμα με συναρτήσεις επεξεργασίας συμβολοσειρών 25
2.5 Ένα παράδειγμα με συναρτήσεις για το χειρισμό της ώρας και ημερομηνίας 26
2.6 Ένα παράδειγμα για τη χρήση συνάρτησης 27
2.7 Μετατροπέας δολαρίων σε ευρώ με χρήση συνάρτησης 28
2.8 Ένα παράδειγμα κλήσης συνάρτησης κατά τιμή 30
2.9 Ένα παράδειγμα κλήσης συνάρτησης κατά αναφορά 31
2.10 Ένα παράδειγμα υπερφόρτωσης συνάρτησης 33
3.1 Ένα απλό παράδειγμα με κλάσεις 38
3.2 Ένα απλό παράδειγμα με κλάσεις και κατασκευαστές 40
3.3 Κλάσεις και συναρτήσεις καταστροφής 42
3.4 Ένα ολοκληρωμένο παράδειγμα με κλάσεις 42
3.5 Στατικά δεδομένα μέλη 45
3.6 Ο δείκτης this 45
3.7 Αντικείμενα ως ορίσματα συναρτήσεων μελών 48
3.8 Συναρτήσεις μέλη που επιστρέφουν αντικείμενα 50
3.9 Συμβολοσειρές ως δεδομένα μέλη κλάσεων 52
3.10 C++ συμβολοσειρές ως δεδομένα μέλη κλάσεων 53

3.11 Πίνακες από αντικείμενα 54
3.12 Δείκτες σε αντικείμενα και κλήση κατά αναφορά 57
3.13 Κλάσεις ως δεδομένα μέλη κλάσεων 59
4.1 Υπερφόρτωση Μονομελών Τελεστών 66
4.2 Υπερφόρτωση Δυμελών Τελεστών 68
4.3 Συναρτήσεις μετατροπής 71
4.4 Φίλες Συναρτήσεις ως γέφυρα δύο μη συσχετιζόμενων Κλάσεων 73
4.5 Φίλες Συναρτήσεις για πιο ευέλικτες συναρτήσεις υπερφόρτωσης 74
5.1 Κατασκευαστές Παράγωγης Κλάσης 81
5.2 Υπερκάλυψη συναρτήσεων μελών 84
5.3 Ένα ολοκληρωμένο παράδειγμα δημόσιας κληρονομικότητας 85
5.4 Ιδεατές Συναρτήσεις 89
A.1 Παραδείγματα με χειριστές 100

Κεφάλαιο 1

Βασικές Έννοιες

ντιρεκτίβες - μεταβλητές - εμβέλεια μεταβλητών - τύποι δεδομένων - πίνακες - δείκτες - αριθμητικοί τελεστές - συσχετιστικοί τελεστές - λογικοί τελεστές - εκφράσεις - δηλώσεις - συναρτήσεις - είσοδος - έξοδος - βρόχοι - αποφάσεις

Το πρώτο πρόγραμμα σε C++

Το πρώτο μας πρόγραμμα σε C++ τυπώνει στην οθόνη την πρόταση ``Hello World!".

Πρόγραμμα 1.1: Hello World!

```
1 #include <iostream>  // preprocessor directive
2 using namespace std; // using directive
3
4 int main()
5 {
6    cout << "Hello World!\n";
7    return 0;
8 }
```

Ντιρεκτίβες
Οι δύο πρώτες γραμμές του κώδικα περιέχουν ντιρεκτίβες (directives), δλδ. εντολές για τον μεταγλωττιστή

Οι προ- επεξεργαστή ντιρεκτίβες ξεκινούν με το σύμβολο #, και δίνουν την εντολή στον μεταγλωττιστή να εισάγει αρχεία πηγαίου κώδικα πριν τη μεταγλώττιση. Για παράδειγμα, το `iostream` αρχείο επικεφαλίδας[1], που ορίζει αντικείμενα όπως το `cout` και το `<<` τελεστή εισαγωγής.

Οι using ντιρεκτίβες δηλώνουν το χώρο ονομάτων (namespace) στον οποίο ανήκουν στοιχεία που περιέχονται στο πρόγραμμά μας. Για παράδειγμα, το `cout` αντικείμενο ορίζεται στο `std` χώρο ονομάτων. Αν δε θέλουμε να χρησιμοποιήσουμε την using ντιρεκτίβα, θα πρέπει κάθε φορά να δηλώνουμε το όνομα του χώρου πριν από κάθε τέτοιο στοιχείο, π.χ.:
`std::cout << "Hello World!\n";`

[1] Οι νεότερες εκδόσεις των αρχείων επικεφαλίδων (header files) της Καθιερωμένης C++ δεν περιέχουν επέκταση.

Συναρτήσεις
Κάθε πρόγραμμά μας στην C++ πρέπει να έχει μία `main` συνάρτηση, από την οποία ξεκινά η εκτέλεσή του. Στο πρόγραμμά μας, δεν δέχεται ορίσματα `()` και επιστρέφει μια ακέραια τιμή. Κάθε μή μηδενική τιμή που επιστρέφει, σημαίνει αποτυχία στην εκτέλεσή του.

Προτάσεις
Η `main` συνάρτηση περιέχει δύο προτάσεις (statements). Οι προτάσεις τελειώνουν πάντα με ελληνικό ερωτηματικό (`;`) και είναι εντολές προς τον υπολογιστή προκειμένου να κάνει κάτι.

cout
Το `cout` αντικείμενο αντιπροσωπεύει το καθιερωμένο ρεύμα εξόδου, το οποίο κανονικά είναι η οθόνη μας. Ο << τελεστής εισαγωγής (insertion operator) κατευθύνει το περιεχόμενο των μεταβλητών που βρίσκονται δεξιά του προς το αντικείμενο που βρίσκεται αριστερά. Συνεπώς, η πρώτη πρόταση (γραμμή 6) τυπώνει στην οθόνη του υπολογιστή μας ``Hello World!''.

return
Η δεύτερη πρόταση (γραμμή 7) δίνει την εντολή στην `main` να επιστρέψει την τιμή `0` σε αυτόν που την κάλεσε (το λειτουργικό σύστημα ή ο μεταγλωττιστής).

Σχόλια
Σε ένα πρόγραμμα στη C++ τα σχόλια ξεκινούν με το σύμβολο `//` και τελειώνουν στο τέλος της γραμμής. Ένας δεύτερος τρόπος, συνήθως για μεγάλα σχόλια, είναι χρησιμοποιώντας τα `/*` για να δηλώσουμε την αρχή σχολίου και `*/` για το τέλος, π.χ.

```
/* Αυτό είναι ένα
μεγαλύτερο σχόλιο
*/
```

Μεταγλώττιση και εκτέλεση προγράμματος

Για να μεταγλωττίσουμε το παραπάνω πρόγραμμα στο περιβάλλον του εργαστηρίου (Windows XP, Visual Studio C++ 2006) δημιουργούμε ένα καινούριο C++ Source File αρχείο με την επέκταση .cpp. Στη συνέχεια το μεταγλωττίζουμε εκτελώντας διαδοχικά Build → compile, Build → Build και το εκτελούμε Build → Execute.

Για να μεταγλωττιστεί σε Linux περιβάλλον χρησιμοποιούμε την εντολή g++, ή μέσω ενός ide της προτίμησής μας (π.χ. MonoDevelop, Eclipse CDT...).

Μεταβλητές και βασικοί τύποι

Στη C++ μπορούμε να δηλώσουμε μεταβλητές σε οποιοδήποτε σημείο μπορεί να δοθεί μια έκφραση. Μια δήλωση (declaration) μιας μεταβλητής, γνωστοποιεί στο πρόγραμμα το όνομα της μεταβλητής και τον τύπο της, ενώ παράλληλα δεσμεύει μνήμη με το αντίστοιχο μέγεθος και τύπο.

Η C++ έχει ένα σύνολο από θεμελιώδεις τύπους για την αποθήκευση δεδομένων. Ο πίνακας 1.1 περιέχει μερικούς βασικούς τύπους, μαζί με το μέγεθος, τον αριθμό των δεκαδικών ψηφίων (όπου ορίζονται) και τη μνήμη που δεσμεύουν σε ένα 32-bit περιβάλλον (όπως είναι αυτό των εργαστηρίων)[2].

[2]Για να βρει κανείς το μέγεθος ενός τύπου ή μιας παράστασης μπορεί να χρησιμοποιήσει το τελεστή `sizeof`, π.χ. η `cout << sizeof(int);` επιστρέφει το μέγεθος του ακεραίου `int`.

Τύπος		Μέγεθος		Δεκαδικά	Μνήμη (Bytes)
		Από	Έως		
χαρακτήρας	`char`	-128	127	-	1
λογικός	`bool`	`true` ή `false`		-	1
ακέραιος	`short`	-32.768	32.767	-	2
	`int`	$-2.147.483.648$	$-2.147.483.647$	-	4
	`long`	$-2.147.483.648$	$-2.147.483.647$	-	4
κινητής υποδιαστολής	`float`	$3,4 \times 10^{-38}$	$3,4 \times 10^{38}$	7	4
	`double`	$1,7 \times 10^{-308}$	$1,7 \times 10^{308}$	15	8
	`long double`	$3,4 \times 10^{-4932}$	$3,4 \times 10^{4932}$	19	10

Πίνακας 1.1: Βασικοί τύποι δεδομένων στη C++

Το παρακάτω πρόγραμμα 1.2 δέχεται μια τιμή από το πληκτρολόγιο σε δολάρια και την μετατρέπει σε ευρώ.

Πρόγραμμα 1.2: Μετατροπέας δολαρίων σε ευρώ

```
1  // currency.cpp
2  // demonstrates some C++ basic types, const, cin
3  #include <iostream>
4  using namespace std;
5
6  int main() {
7     const float EU = 0.73078;  // 1 us dollar is 0.73078 euro
8     double usvalue;   // value in us dollars
9     cout << "Enter value in us dollars: ";
10    cin >> usvalue;
11    float euvalue = usvalue * EU; // value in euro
12    cout << "Equivalent value in euro: " << euvalue << endl;
13    return 0;
14 }
```

const

Το `const` διευκρινιστικό (qualifier) προηγείται από τον τύπο και διευκρινίζει πως η τιμή της μεταβλητής δεν θα αλλάξει κατά τη διάρκεια του προγράμματος. Κάθε προσπάθεια αλλαγής θα καταλήγει σε λάθος στον μεταγλωττιστή. Για παράδειγμα, θα ήταν να λάθος να πούμε:

```
1  EU = 0.83234;  // error: assignment of read-only variable 'EU'
```

cin

Το `cin` αντικείμενο αντιπροσωπεύει το καθιερωμένο ρεύμα εισόδου, το οποίο κανονικά είναι το πληκτρολόγιο. Το πρόγραμμα σταματά την εκτέλεσή του προκειμένου ο χρήστης να πληκτρολογήσει κάποια τιμή και στη συνέχεια το πλήκτρο Enter. Ο `>>` τελεστής εξαγωγής (extraction operator) τοποθετεί τη τιμή που έχουμε πληκτρολογήσει στη μεταβλητή που βρίσκεται δεξιά του.

Συνεπώς, το πρόγραμμα 1.2 έχει σαν έξοδο:

```
Enter value in us dollars: 20
```

```
Equivalent value in euro: 14.6156
```

Απρόσημοι τύποι
Ο πίνακας 1.1 δείχνει το μέγεθος των μεταβλητών που υποστηρίζει κάθε τύπος. Αν δεν μας ενδιαφέρει το πρόσημο ενός χαρακτήρα ή ακεραίου μπορούμε να αλλάξουμε το όριό του ώστε να ξεκινά από το μηδέν, διπλασιάζοντας έτσι το θετικό του μέγεθος. Αυτό το κάνουμε χρησιμοποιώντας το διευκρινιστικό `unsigned` (απρόσημος). Για παράδειγμα, ο απρόσημος ακέραιος

```
1  unsigned int counter;
```

έχει όριο τιμών από 0 έως 4.294.967.295.

Πίνακες

Μπορούμε να ορίσουμε μια ομάδα μεταβλητών ιδίου τύπου ως πίνακα (array).

```
1  int numbers[4];   // an array of 4 integers
2  char name[20]; // an array of 20 characters
```

Μπορουμε να αρχικοποιήσουμε τους πίνακες καθώς τους ορίζουμε:

```
1  char name[] = "John";   // an array of 5 characters (count the last '\0')
2  float grades[4] = {5.2, 9, 7.3, 10};   // an array of 4 floats
```

Κάθε που ορίζουμε έναν δείκτη δεσμεύουμε τόσες διαδοχικές θέσεις μνήμης όσο το μέγεθος του πίνακα.

Αναφερόμαστε σε στοιχείο του πίνακα διευκρινίζοντας τη θέση του στοιχείου μέσα σε αγκίλες. Οπως και στη C, η αρίθμιση των στοιχείων ξεκινά από το 0.

```
1  int numbers[5] = {9, 1, 3, 4, 2};
2  cout << numbers[2];  // prints 3
3  numbers[2] = 8;      // 3 becomes 8
4  cout << numbers[2];  // prints 8
```

Αναφορές

Η αναφορά (reference, &) είναι ένα εναλλακτικό όνομα για μια μεταβλητή.

Για παραδειγμα, η παρακάτω αναφορά σε ακέραιο `r`.

```
1  int i = 10;
2  int& r = i; // r, i reference to the same int, that is 10
3  r = 20;     // i = 20
```

Διεύθυνση	Τιμή	Μεταβλητή
0		
...		
0x7fff01889624		
...		
0x7fff0188962b	0x7fff0188962f	p
...		
0x7fff0188962f	10	a
...		

Πίνακας 1.2: Αναπαράσταση της μνήμης

Δείκτες

Η μνήμη του ηλεκτρονικού υπολογιστή οργανώνεται σε μια σειρά από bytes (θέσεις μνήμης), καθένα από το οποία χαρακτηρίζεται από έναν μοναδικό ακέραιο αριθμό, μια διεύθυνση (address).

Κάθε μεταβλητή του προγραμματός μας καταλαμβάνει χώρο στη μνήμη ανάλογο με τον τύπο του (π.χ. σύμφωνα με τον Πίνακα 1.1 ένας ακέραιος καταλαμβάνει 4 συνεχόμενες θέσεις μνήμης, bytes). Χρησιμοποιώντας τον τελεστή διεύθυνσης & έχουμε πρόσβαση στην διεύθυνση της πρώτης θέσης (του πρώτου byte) μιας μεταβλητής. Π.χ. ο παρακάτω κώδικας ορίζει έναν ακεραιο a και τυπώνει την διεύθυνσή του &a:

```
1 int a;
2 cout << "The address of a is: " << &a << endl;
```

Επειδή οι τιμές των διευθύνσεων τείνουν να γίνονται πολύ μεγάλες, στην έξοδο για συντομία τυπώνονται σε δεκαεξαδική μορφή (0x:segment:offset):

```
The address of a is 0x7fffe9c0499c
```

Οι δείκτες (pointers) είναι ειδικού τύπου μεταβλητές που μπορούν να περιέχουν διευθύνσεις μνήμης άλλων μεταβλητών. Οι δείκτες ορίζονται χρησιμοποιώντας τον τύπο της μεταβλητής στην οποία θέλουμε να δείχνουνε μαζί με τον τελεστή *. Π.χ. ο δείκτης p είναι δείκτης ακεραίου:

```
1 int *p;
```

Στο παρακάτω παράδειγμα αναθέτουμε στο δείκτη p τη διεύθυνση της ακέραιας μεταβλητής a:

```
1 int a = 10;
2 int *p;
3 p = &a;  // p points to a
```

Αν θέλαμε να απεικονίσουμε την κατάσταση της μνήμης θα ήταν κάτι σαν τον Πίνακα 1.2.

Για να τυπώσουμε την τιμή που περιέχεται εκεί που δείχνει ένας δείκτης χρησιμοποιούμε τον τελεστή *:

```
1 int a = 10;
2 int *p;
3 p = &a;     // p points to a
```

```
4  cout << *p; // prints 10
```

Γενικότερα, οι δείκτες επιτρέπουν κάθε λειτουργία που μπορούμε να εκτελέσουμε άμμεσα σε μια μεταβλητή.

Παραδείγματα χρήσης δεικτών

Έμμεση αναφορά
Μπορούμε να χρησιμοποιήσουμε τους δείκτες για να προσπελάσουμε έμμεσα μεταβλητές, να πάρουμε τις τιμές τους ακόμα και να τις αλλάξουμε (indirection).

Το παρακάτω παράδειγμα αλλάζει την τιμή της μεταβλητής `a` μέσω του δείκτη `p`:

```
1  int a = 10;
2  int *p = &a;
3
4  *p = 5;      // alter the value of a through p
5  cout << a;  // prints 5
```

Πίνακες και δείκτες
Το όνομα ενός πίνακα μπορεί να θεωρηθεί σαν ένας δείκτης στο πρώτο στοιχείο του πίνακα. Παρακάτω ο `p` δείχνει στο `numbers[0]`:

```
1  int numbers[10];
2  int *p;
3  p = numbers;   // equals p = &numbers[0];
```

Επιπλέον η εκφραση `numbers[i]` είναι ισοδύναμη με την `*(numbers + i)` (όπου `i` ακέραιος), γι' αυτό και οι παρακάτω προτάσεις είναι ισοδύναμες:

```
1  int i;
2  ...
3  *numbers = 4;      // equals numbers[0] = 4;
4  *(numbers + i) = 5;  // equals numbers[i] = 5;
```

Κάθε που αυξάνουμε έναν δείκτη κατά ένα συγκεκριμένο αριθμό `n` (π.χ. `p++`) είναι σαν να του λέμε να δείξει στο στοιχείο της μνήμης που βρίσκεται `n` θέσεις μετά. Εχει νόημα να κάνουμε κάτι τέτοιο όταν γνωρίζουμε το περιεχόμενο των επόμενων θέσεων, π.χ. στους πίνακες:

```
1  int numbers[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
2  int *p;
3  p = numbers;
4  cout << *p; // prints 1
5  ++p;
6  cout << *p; // prints 2
```

Αλλες χρήσεις

Πέρα από τη χρηση των δεικτών ως τρόπο πρόσβασης σε μεταβλητές, αντικείμενα, πίνακες, οι δείκτες χρησιμοποιούνται στη δημιουργια δομών δεδομένων (όπως οι συνδεδεμένες λίστες, δυαδικά δέντρα...). Επιπλέον, στη C++ οι δείκτες έχουν εφαρμογή στις ιδεατές συναρτήσεις (Κεφάλαιο 5), στο τελεστή new, στον `this` δείκτη των αντικειμένων....

Περισσότερα για τους δείκτες στο Κεφάλαιο **??**.

Μετατροπές

Η C++ επιτρέπει εκφράσεις στην οποία συμμετέχουν δεδομένα διάφορων τύπων. Στο παρακάτω παράδειγμα:

Πρόγραμμα 1.3: Αυτόματη Μετατροπή Τύπων

```
1  // conversion.cpp
2  // demonstrates data conversions
3  #include <iostream>
4
5  using namespace std;
6
7  int main() {
8     int count = 7;
9     float weight = 150.5;
10
11    double totalweight = count * weight;
12    cout << "Total weight: " << totalweight << endl;
13    return 0;
14 }
```

το συνολικό βάρος (`totalweight`) είναι τύπου `double` και υπολογίζεται από τον πολλαπλασιασμό ενός `int` με έναν `float`. Προκειμένου να γίνει ο υπολογισμός αυτός, ο μεταγλωττιστής πραγματοποιεί εσωτερικά αντίστοιχες μετατροπές: ο πιο ``μικρός'' τύπος μετατρέπεται στον πιο ``μεγάλο''.

char	short	int	long int	float	double	long double
μικρότερος						μεγαλύτερος

Πίνακας 1.3: Διάταξη τύπων δεδομένων στη C++

Ετσι στο πρόγραμμα 1.3 (γραμμή 11) η ακέραια τιμή `count` μετατρέπεται σε `float` και αποθηκεύεται προσωρινά πριν γίνει ο πολλαπλασιασμός. Στη συνέχεια το αποτέλεσμα, που είναι ακόμα τύπου `float`, μετατρέπεται σε `double` και αναθέτεται στην μεταβλητή `totalweight`.

Υπάρχουν περιπτώσεις που τέτοιες μετατροπές δεν γίνονται αυτόματα ή χωρίς κάποια προειδοποίηση (warning) από το μεταγλωττιστή. Σε τέτοιες περιπτώσεις για να αποφύγουμε απροσδιόριστα αποτελέσματα, η C++ μας επιτρέπει να δηλώσουμε ρητά μια μετατροπή (cast). Ενας τρόπος για να το κάνουμε αυτό είναι η χρήση της `static_cast` (στατική μετατροπή)[3]. Έτσι, στο παρακάτω παράδειγμα:

[3]Στη C++ υπάρχουν τέσσερα είδη μετατροπών: στατικές (static casts), δυναμικές (dynamic casts), reinterpret casts και σταθερές μετατροπές (const casts). Στα πλαίσια του εργαστηρίου θα δούμε μόνο τις πρώτες.

```
1  int intVar = 1500000000;   //1.500.000.000
2  intVar = (static_cast<double>(intVar) * 10 ) / 10;
```

η `intVar` μετατρέπεται προσωρινά σε `double`, για να χωρέσει το αποτέλεσμα του υπολογισμού `intVar*10`.

Αν δεν κάναμε ρητά τη μετατροπή, το αποτέλεσμα που θα παίρναμε θα ήταν απροσδιόριστο.

Σημειώστε πως μετατροπές από έναν τύπο σε έναν άλλο κάνουμε μόνο όταν είναι απαραίτητο.

Καθιερωμένοι χειριστές

Οι καθιερωμένοι χειριστές (manipulators) είναι τελεστές που χρησιμοποιούνται ανάμεσα από τελεστές εισαγωγής << προκειμένου να τροποποιήσουν το πώς θα εμφανίζονται τα δεδομένα.

Για παράδειγμα, η `setw` ορίζει το πλάτος της εκτύπωσης που ακολουθεί.

Πρόγραμμα 1.4: Εκτύπωση βαθμολογίας

```
1  // setw.cpp
2  // demonstrates setw manipulator
3  #include <iostream>
4  #include <iomanip>
5
6  using namespace std;
7
8  int main() {
9     int grade1 = 10, grade2 = 4, grade3 = 7;
10
11    cout  << setw(4) << "Name" << setw(6) << "Grade" <<endl
12       << setw(4) << "John" << setw(6) << grade1 <<endl
13       << setw(4) << "Ross" << setw(6) << grade2 <<endl
14       << setw(4) << "Jane" << setw(6) << grade3 <<endl;
15    return 0;
16 }
```

Έτσι, ο παραπάνω κώδικας τυπώνει:

```
Name Grade
John    10
Ross     4
Jane     7
```

endl
Ο `endl` χειριστής στέλνει στην έξοδο τον χαρακτήρα αλλαγής γραμμής `'\n'` και παράλληλα αδειάζει (flush) την περιοχή προσωρινής αποθήκευσης του ρεύματος εξόδου (buffer output).

Το Παράρτημα Α αναφέρει κάποιους βασικούς χειριστές.

Τελεστές

Αριθμητικοί Τελεστές

Οι τελεστές +, −, *, /, και % (arithmetic operators) μας επιτρέπουν να εκτελούμε αριθμητικές πράξεις.

```
1  cout << 2 + 3 << endl      // 5
2       << (4 + 2) - 5 << endl   // 1
3       << (2 * 8) / 4 << endl   // 4
4       << 6 % 8 << endl      // 6
5       << 10 % 8 << endl;    // 2
```

Οι αριθμητικοί τελεστές ανάθεσης, όπως είναι οι +=, −=, *=, /=, %=, >>=, <<=, ^= , και |=, μας επιτρέπουν να γράφουμε πιο σύντομες και κομψές εκφράσεις (compound assignment):

```
1  int total = 10;
2  int item = 1;
3  total += item; // total = total + item; total becomes 11
4  total -= item; // total = total - item; total becomes 10
```

Οι ++, −− αυξάνουν ή μειώνουν αντίστοιχα κατά ένα τη τιμή μιας μεταβλητής. Όταν τοποθετούνται πριν το όνομα μιας μεταβλητής (prefix) η μεταβλητή πρώτα μεταβάλλεται και έπειτα χρησιμοποιείται, ενώ όταν τοποθετούνται μετά (prostfix) πρώτα χρησιμοποιείται και μετά μεταβάλλεται

```
1  int item = 10;
2  cout << item << endl // 10
3       << ++item << endl  // 11
4       << item << endl // 11
5       << item++ << endl  // 11
6       << item << endl;   // 12
```

Συσχετιστικοί Τελεστές

Οι συσχετιστικοί τελεστές (relational and equality operators) μας επιτρέπουν να συγκρίνουμε δύο τιμές.

Τελεστής	Συσχέτιση
==	Ίσο
!=	Διάφορο
>	Μεγαλύτερο
<	Μικρότερο
>=	Μεγαλύτερο ή ίσο
<=	Μικρότερο ή ίσο

Πίνακας 1.4: Συσχετιστικοί Τελεστές στη C++

Το αποτέλεσμα της κάθε σύγκρισης είναι 1 (true) αν η σύγκριση είναι αληθής και 0 (false) αν είναι ψευδής.

```
1  cout << (10 < 2) << endl   // 0
2       << (20 != 2) << endl; // 1
```

Λογικοί τελεστές

Τα λογικά αποτελέσματα στη C++ μπορούν να συνδυαστούν με τη χρήση των παρακάτω λογικών τελεστών (logical operators):

Τελεστής	Συσχέτιση
!	άρνηση - όχι
||	διάζευξη - ή
&&	σύζευξη - και

Πίνακας 1.5: Λογικοί Τελεστές στη C++

```
1  cout << ((10 < 12) && (10 > 8)) << endl   // 1
2       << ((20 != 2) || (3 == 2)) << endl   // 1
3       << (!(2 < 3)) << endl;       // 0
```

Βρόχοι

Οι βρόχοι (loops) μας επιτρέπουν να επαναλαμβάνουμε ένα συγκεκριμένο κομμάτι του κώδικα μέχρι να πάψει να ισχύει μια συνθήκη. Στη C++ μπορούμε να ορίσουμε βρόχους χρησιμοποιώντας τις εντολές `for`, `while`, ή `do`.

for
Η εντολή `for` χρησιμοποιείται ως εξής:

```
for(αρχική_έκφραση; έκφραση_ελέγχου; τελική_έκφραση) {
        πρόταση;
        πρόταση;
        ...
}
```

Οποιαδήποτε έκφραση μπορεί να παραληφθεί.

Για παράδειγμα, ο παρακάτω κώδικας αυξάνει την `initialterm` κατά `5` και την τυπώνει, για όσο το `i` είναι μικρότερο του `10`. Η αρχική τιμή του `i` είναι `0` και σε κάθε επανάληψη αυξάνεται κατά `1`:

Πρόγραμμα 1.5: For βρόχος

```
1  // simplefor.cpp
2  // demonstrates 'for' loops
3  #include <iostream>
```

```
4  using namespace std;
5
6  int main() {
7     int difference = 5;
8     int initialterm = 0 ;
9     for(int i = 0; i < 10; i++) {
10       cout << initialterm << '\t';
11       initialterm += difference;
12    }
13    cout << endl;
14    return 0;
15 }
```

Συνεπώς, ο παραπάνω κώδικας τυπώνει:

```
0    5    10    15    20    25    30    35    40    45
```

while
Η εντολή `while` χρησιμοποιείται ως εξής:

```
while(έκφραση_ελέγχου) {
        πρόταση;
        πρόταση;
        ...
}
```

Το παρακάτω πρόγραμμα δεν σταματά να εκτελείται, εκτός όταν ο χρήστης εισάγει 'y' (yes).

Πρόγραμμα 1.6: While βρόχος

```
1  // simplewhile.cpp
2  // demonstrates 'while' loops
3  #include <iostream>
4  using namespace std;
5
6  int main() {
7     char ch = '';  // be sure that ch isn' t initialized to 'y'
8     while (ch != 'y') {
9        cout << "Do you want to exit (y/n)? ";
10       cin >> ch;
11    }
12    return 0;
13 }
```

```
Do you want to exit (y/n)? n
Do you want to exit (y/n)? n
Do you want to exit (y/n)? y
```

do
Η εντολή do χρησιμοποιείται ως εξής:

```
do {
        πρόταση;
        πρόταση;
        ...
} while (έκφραση_ελέγχου);
```

Σημειώστε πως η while ελέγχει την συνθήκη στην αρχή του βρόχου, ενώ η do στο τέλος. Συνεπώς η do πάντα θα εκτελεί τις εντολές του βρόχου τουλάχιστον μία φορά.

Αποφάσεις

Η C++ μας επιτρέπει να εκτελούμε ένα σύνολο εντολών υπό συνθήκη χρησιμοποιώντας τις εντολές απόφασης if, if...else και switch.

if
Η εντολή if χρησιμοποιείται ως εξής:

```
if (έκφραση_ελέγχου) {
        πρόταση;
        ...
}
```

if...else
Την εντολή if μπορεί να ακολουθηθεί η εντολή else στην οποία προσδιορίσουμε εντολές που θα εκτελεστούν αν η συνθήκη δεν ισχύει.

```
if (έκφραση_ελέγχου) {
        πρόταση;
        ...
} else {
        πρόταση;
        ...
}
```

Για παράδειγμα, το παρακάτω πρόγραμμα ενημερώνει το χρήστη για το αν ο ακέραιος που εισήγαγε είναι άρτιος/ζυγός (even) ή περιττός/μονός (odd).

Πρόγραμμα 1.7: if...else

```
1  // simpleif.cpp
2  // demonstrates if..else statements
3
4  #include <iostream>
5  using namespace std;
6
7  int main() {
```

```
 8     int num = 0;
 9     cout << "Enter a number: ";
10     cin >> num;
11
12     if ((num % 2) == 0) {
13        cout << num << " is even." << endl;
14     } else {
15        cout << num << " is odd." << endl;
16     }
17     return 0;
18  }
```

```
Enter a number: 4
4 is even.
Enter a number: 5
5 is odd.
```

Μπορούμε να εμφωλεύουμε επιπλέον `if` και `if...else` αποφάσεις ανάμεσα στις εντολές που ακολουθούν τις `if` και `else`. Δώστε όμως προσοχή στις παρενθέσεις, ώστε να ομαδοποιείτε τις εντολές με τον τρόπο που επιθυμείτε.

switch

Αν οι αποφάσεις μας στηρίζονται στην τιμή που έχει μία μεταβλητή, μπορούμε να χρησιμοποιήσουμε την εντολή `switch` με τον εξής τρόπο:

```
switch (n) {                        // n is a character or an integer
        case static1:               // a value
                πρόταση;
                ...
                break;              // break out of switch
        case static2:               // another value
                πρόταση;
                ...
                break;
        ...
default:
                πρόταση;
                ...
}
```

Το παρακάτω πρόγραμμα ελέγχει το χαρακτήρα που εισήγαγε ο χρήστης.

Πρόγραμμα 1.8: Switch...case

```
1  // simpleswitch.cpp
2  // demonstrates switch statements
3
4  #include <iostream>
5  using namespace std;
```

```
6
7  int main() {
8     char ch;
9     cout << "Are you sure? (y/n) ";
10    cin >> ch;
11
12    switch (ch) {
13       case 'y':
14          cout << "You are sure." << endl;
15          break;
16       case 'n':
17          cout << "You are not sure." << endl;
18          break;
19       default:
20          cout << "You are something else..." << endl;
21    }
22    return 0;
23 }
```

```
Are you sure? (y/n) y
You are sure.
```

break

Η εντολή `break` μας εξάγει από το βρόχο στον οποίο ανήκει, όπως μας εξάγει από μια `switch`. Ο παρακάτω ατέρμονος βρόχος `for` θα σταματήσει να εκτελείται όταν ο χρήστης πληκτρολογήσει το χαρακτήρα 'a'.

```
1  char ch;
2  int position = 1;
3  for (;;) {
4     cin >> ch;
5     if (ch == 'a') {
6        cout << "The " << position << "th character is alfa." << endl;
7        break;
8     }
9     ++position;
10 }
```

continue

Η εντολή `continue` μας επιτρέπει να συνεχίσουμε την εκτέλεση ενός βρόχου από την αρχή του, παραλείποντας τις εντολές που την ακολουθούν.

Ο παρακάτω κώδικας αθροίζει μόνο του άρτιους αριθμούς που εισάγει ο χρήστης.

```
1  int sumOfEven = 0;
2  int num = -1;  // be sure that num isn' t initialized to 0
3  while (num != 0 ) {
4     cin >>  num;
5     if (num % 2 != 0)
```

```
6         continue;
7      sumOfEven += num;
8  }
9  cout << sumOfEven;
```

Σύντομη έκφραση ελέγχου
Επιπλέον, υπάρχει μια σύντομη έκφραση ελέγχου χρησιμοποιώντας τους τελεστές ? (conditional operator) και : και τρεις τελεσταίους:

```
(έκφραση_ελέγχου) ?  έκφραση1 : έκφραση2;
```

Αν η έκφραση_ελέγχου είναι αληθής, τότε εκτελείται η έκφραση1, αλλιώς εκτελείται η έκφραση2, π.χ.

```
1  int a = 10, b = 30;
2  int c = (a > b) ? a : b;    // c becomes 30
```

Ασκήσεις

1.1 Δημιουργείστε σε C++ ένα κομπιουτεράκι που να υλοποιεί τις τέσσερις βασικές πράξεις: πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση. Ζητήστε από το χρήστη δύο αριθμούς και το είδος της πράξης. Τυπώστε το αποτέλεσμα της πράξης.

1.2 Γράψτε σε C++ ένα πρόγραμμα που με την χρήση της `for` να εκτυπώνεται το παρακάτω σχήμα.

```
     *
    ***
   *****
  *******
 *********
***********
```

1.3 Γράψτε σε C++ ένα πρόγραμμα που να υπολογίζει το παραγωντικό κάθε αριθμού που εισάγει ο χρήστης. Το πρόγραμμα να τερματίζει όταν χρήσης εισάγει τον αριθμό μηδέν.

1.4 Δημιουργείστε σε C++ ένα κομπιουτεράκι που να υλοποιεί τέσσερις βασικές πράξεις μεταξύ κλασμάτων: πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση. Ζητήστε από το χρήστη να εισάγει τον αριθμιτή και παρονομαστή του πρώτου κλάσματος, επειτα του δεύτερου κλάσματος και τέλος το είδος της πράξης. Τυπώστε το αποτέλεσμα της πράξης.

1.5 Γράψτε σε C++ ένα πρόγραμμα που να διαβάζει μια γραμμή από χαρακτήρες. Στο τέλος να τυπώνει τον αριθμό των πεζών χαρακτήρων (a-z), των κεφαλαίων χαρακτήρων (A-Z), των ψηφίων (0-9), και τον αριθμό των υπόλοιπων συμβόλων που εισήγαγε ο χρήστης. (Θυμηθείτε πως οι πεζοί χαρακτήρες είναι αυτοί που ικανοποιούν την παράσταση `ch >= 'a' && ch <= 'z'`)

1.6 Γράψτε σε C++ ένα πρόγραμμα που να τυπώνει τη παρακάτω διάταξη. Ζητήστε από το χρήστη τον αριθμό των γραμμών.

```
5
5   4
5   4   3
5   4   3   2
5   4   3   2   1
```

Κεφάλαιο 2

Δομές και Συναρτήσεις

δομές - απαριθμήσεις - συναρτήσεις - κλήση κατά τιμή - κλήση κατά αναφορά - υπερφόρτωση συναρτήσεων - πολυμορφισμός

Δομές

Μια δομή (structure) είναι ένας καθορισμένος από το χρήστη τύπος δεδομένων. Αποτελεί μια συλλογή δεδομένων οποιουδήποτε τύπου.

Ορίζουμε μία δομή χρησιμοποιώντας το διευκρινιστικό `struct` ακολουθουμενο από το όνομα της δομής. Το σώμα της δομής περικλείεται από αγκύλες . Στο τέλος τοποθετούμε ελληνικό ερωτηματικό (;). Για παράδειγμα, η παρακάτω δομή `student` ορίζει ένα νέο τύπο που ονομάζεται `student` που αποτελείται από δύο μέλη: έναν ακέραιο και έναν δεκαδικό.

```
1  struct student {
2     int studentId;
3     float grade;
4  };
```

Ορίσουμε μια μεταβλητή τύπου `student` ως εξής:

```
1  student john;  // john variable has type student
```

Χρησιμοποιώντας τον τελεστή τελεία (`.`), έχουμε πρόσβαση στα μέλη της δομής:

```
1  john.studentId = 123456;   // gives value to john's studentId
```

Ο παρακάτω κώδικας ορίζει δύο `student` και τυπώνει τα στοιχεία τους:

Πρόγραμμα 2.1: Ένα απλό παράδειγμα με δομές

```
1  // structstudent.cpp
```

```
2  // demonstrates structures
3  #include <iostream>
4  using namespace std;
5
6  struct student {
7     int studentId;
8     float grade;
9  };
10
11 int main() {
12
13    student john;  // define a structure variable
14    john.studentId = 12345; // populate john
15    john.grade = 8.8;
16
17    student mary = {12346, 7.5};  // define another structure variable
18
19    // print structures
20    cout << "John has studentId: " << john.studentId
21       << ", and grade: " << john.grade << endl;
22    cout << "Mary has studentId: " << mary.studentId
23       << ", and grade: " << mary.grade << endl;
24
25    return 0;
26 }
```

Το πρόγραμμα 2.1 έχει σαν έξοδο:

```
John has studentId: 12345, and grade: 8.8
Mary has studentId: 12346, and grade: 7.5
```

Ο τελεστής ανάθεσης τιμής = (assignment operator), για τις δομές δουλεύει όπως θα περιμέναμε:

```
1  john = mary; /* means that
2        john.studentId = mary.studentId
3        john.grade = mary.grade */
```

Αντίθετα, άλλοι τελεστές όπως αυτοί τις σύγκρισης (==, > ...), ή των αριθμητικών πράξεων (+, –, ...) έχουν απροσδιόριστα αποτελέσματα ή δεν ορίζονται καν.

Δείκτες και δομές

Μπορούμε να ορίσουμε δείκτες να δείχνουν σε μεταβλητές τύπου δομής, π.χ. ο `pjohn`:

```
1  struct student {
2     int studentId;
3     float grade;
```

```
}

student john;
student *pjohn;
```

Ο pjohn δείχνει στην john:

```
pjohn = &john; // pjohn points to john
```

Για να αναφερθούμε στα στοιχεία της δομής μέσω του δείκτη χρησιμοποιούμε τον τελεστή -> (arrow operator), π.χ.:

```
pjohn->studentId = 10;  // john.studentId becomes 10;
```

Πίνακες και δομές

Μπορούμε να ορίσουμε ένα πίνακα από μεταβλητές τύπου μιας δομής. Για παράδειγμα, ο παρακάτω πίνακας students είναι πίνακας 10 μαθητών (student):

```
student students[10];   // an array of 10 students
```

Αναφερόμαστε στα στοιχεία κάθε μαθητή χρησιμοποιώντας τον τελεστή τελεία (.), αφού πρώτα προσδιορίσουμε τη θέση του στον πίνακα, π.χ.

```
students[1].studentId = 10;   // studentId of the 2nd student becomes 10
```

Στο παρακάτω παράδειγμα ορίζουμε μια δομή που να κρατά το όνομα και τον αριθμό των μελών μιας οικογένειας (family). Στη συνέχεια αποθοικεύουμε σε ένα πίνακα στοιχεία από οικογένειες, τον αριθμό των οποίων δηλώνει ο χρήστης. Στο τέλος με τη βοήθεια δείκτη τυπώνουμε το πίνακα αυτό.

Πρόγραμμα 2.2: Ένα παράδειγμα με δομές

```
// families.cpp
#include <iostream>
using namespace std;

struct family {
   char name[20];
   int members;
};

int main() {
   int numOfFamilies;
   cout << "Give number of families: ";
   cin >> numOfFamilies;
```

```
14
15     family families[numOfFamilies]; // an array of families
16     family *pf; // a pointer to a family
17
18     // populate families
19     for(int i=0; i<numOfFamilies; i++) {
20        cout << "Family name: ";
21        cin >> families[i].name;
22        cout << "Number of members: ";
23        cin >> families[i].members;
24     }
25
26     // print all families
27     for(pf=families; pf < &families[numOfFamilies]; pf++) {
28        cout << pf->name << " have "
29           << pf->members << " members" << endl;
30
31     }
32
33     return 0;
34  }
```

```
Give number of families: 3

Family name: jetsons
Number of members: 4
Family name: simpsons
Number of members: 5
Family name: flintstones
Number of members: 3

jetsons have 4 members
simpsons have 5 members
flintstones have 3 members
```

Απαριθμήσεις

Οι μεταβλητές τύπου απαρίθμησης (`enum`) ορίζουν νέους τύπους δεδομένων που περιέχουν μια λίστα από καθορισμένες από το χρήστη τιμές. Η παρακάτω δήλωση, ορίζει τέσσερις ακέραιες σταθερές που ονομάζονται απαριθμητές (enumerators):

```
1  enum season {
2     WINTER, AUTUMN, SUMMER, SPRING
3  }
```

Εξ ορισμού, οι τιμές των απαριθμητών αρχίζουν από το 0 και αυξάνονται βηματικά κατά ένα. Γι' αυτό, `WINTER == 0`, `AUTUMN == 1`, `SUMMER == 2`, `SPRING == 3`. Εκτός αν δηλώσουμε εμείς τις τιμές των απαριθμητών:

```
enum special_character {
   BACKSPACE = '\b',
   NEW_LINE = '\n',
   TAB = '\t'
}
```

Η δήλωση μιας μεταβλητής με τη χρήση απαρίθμησης δίνει μια ιδέα για το τρόπο χρήση της.

```
season s;
   ...
switch (s) {
   case WINTER:
      // do sthg
      break;
   case AUTUMN:
      // do sthg
      break;
   ...
}
```

Εσωτερικά ο μεταγλωττιστής χειρίζεται τους απαριθμητές ως ακέραιους. Δοκιμάστε τι θα τυπώσει το παρακάτω:

```
special_character sc = NEW_LINE;
cout >> sc;
```

Συνήθως θέλουμε να χρησιμοποιήσουμε απαριθμήσεις όταν εκ των προτέρων γνωρίζουμε το πεπερασμένο συνήθως μικρό σύνολο των τιμών ενός τύπου.

Συναρτήσεις

Οι συναρτήσεις ομαδοποιούν τμήματα του κώδικα, που εκτελούν μια συγκεκριμένη δουλειά, κάτω από ένα όνομα. Χρησιμοποιώντας αυτό το όνομα μπορούμε να καλέσουμε την συνάρτηση στο πρόγραμμά μας όσες φορές θέλουμε. Μια συνάρτηση μπορεί να χρειάζεται να γνωρίζει κάποιες τιμές πριν αρχίσει να εκτελεί μια δουλειά. Αυτές οι τιμές γίνονται διαθέσιμες με τη χρήση παραμέτρων (ορίσματα). Μια συνάρτηση μπορεί να επιστρέφει το πολύ μια τιμή.

Οι συναρτήσεις μας επιτρέπουν να κρατάμε οργανωμένο, δομημένο το πρόγραμμά μας. Επιπλέον, χρησιμοποιώντας συναρτήσεις μειώνουμε τις γραμμές κώδικα.

Η βιβλιοθήκη της C++

Η Καθιερωμένη Βιβλιοθήκη της C++ (Standard C++ Library) είναι μια συλλογή από συναρτήσεις, σταθερές, κλάσεις, αντικείμενα και πρότυπα που επιτρέπουν διάφορες λειτουργίες [1].

Οι δηλώσεις των διαφόρων αυτών στοιχείων που παρέχει η βιβλιοθήκη χωρίζονται σε διάφορα αρχεία επικεφαλίδων (header files) τα οποία πρέπει να συμπεριλάβουμε στο πρόγραμμά μας προκειμένου να έχουμε πρόσβαση σε συστατικά του.

Η Καθιερωμένη Βιβλιοθήκη της C++ μπορεί να χωριστεί στις παρακάτω κατηγορίες:

1. C Library: περιλαμβάνει τις βιβλιοθήκες της C. Έτσι επιτυχνάνεται συμβατότητα της C++ με τη C. Σε αυτή τη κατηγορία ανήκουν στοιχεία γενικής χρήσης, μακροεντολές, συναρτήσεις εισόδου/ εξόδους, διαχείρισης μνήμης... Για παράδειγμα, η `cmath` βιβλιοθήκη, στην οποία βρίσκονται οι μαθηματικές συναρτήσεις της C,

Πρόγραμμα 2.3: Ένα παράδειγμα με μαθηματικές συναρτήσεις

```
1  // simplemath.cpp
2  // demonstrates the use of some math functions
3  #include <iostream>
4  #include <cmath>
5
6  using namespace std;
7
8  int main() {
9     // double sqrt (double x);
10    // Returns the square root of x.
11    cout << "The square root of 9 is " << sqrt(9) << endl;
12
13    // double pow (double base, double exponent );
14    // Returns base raised to the power exponent.
15    cout << "5 raised to the 2nd power is " << pow(5, 2) << endl;
16
17    return 0;
18 }
```

η `cstring` με συναρτήσεις επεξεργασίας συμβολοσειρών,

Πρόγραμμα 2.4: Ένα παράδειγμα με συναρτήσεις επεξεργασίας συμβολοσειρών

```
1  // simplestring.cpp
2  // demonstrates the use of some string functions
3  #include <iostream>
4  #include <cstring>
5
6  using namespace std;
7
8  int main() {
9
10    char tmp[40];
```

```
11     // char * strcpy (char *destination, const char *source);
12     // Copies the C string pointed by source into the array pointed
13     // by destination, including the terminating null character
14     strcpy(tmp, "this string was copied.");
15     cout << tmp << endl;
16
17     // char * strcat(char *destination, const char *source );
18     // Appends a copy of the source string to the destination string.
19     char str[80];
20     strcpy(str,"these ");
21     strcat(str,"strings ");
22     strcat(str,"were ");
23     strcat(str,"concatenated.");
24     cout << str << endl;
25
26     // int strcmp(const char *str1, const char *str2);
27     // Compares two strings. Return 0 when both strings are equal
28     char str1[] = "The quick brown dog jumps over the lazy fox";
29     char str2[] = "The quick brown dog jumps over the lazy fox";
30     if (!strcmp(str1, str2))
31        cout << "these strings are equal" << endl;
32
33     // size_t strlen(const char *str);
34     // Returns the length of a string
35     char tmp2[100]="another test string";
36     cout << strlen(tmp) << endl; // prints 19
37
38     return 0;
39 }
```

η ctime με συναρτήσεις για το χειρισμό της ώρας και ημερομηνίας,

Πρόγραμμα 2.5: Ένα παράδειγμα με συναρτήσεις για το χειρισμό της ώρας και ημερομηνίας

```
1  // simpletime.cpp
2  // demonstrates the use of some time functions
3  #include <iostream>
4  #include <ctime>
5
6  using namespace std;
7
8  int main () {
9
10   cout <<  (time(NULL)/ 3600) << " hours since January 1, 1970" << endl;
11
12   return 0;
13 }
```

και άλλες.

2. Miscellaneous libraries: διάφορες βιβλιοθήκες γενικού σκοπού. Π.χ. η `string` βιβλιοθήκη της C++.
3. Standard Template Library - STL: η Καθιερωμένη Βιβλίοθήκη Προτύπων [4]. Βασικά μέρη αυτής της κατηγορίας είναι οι κλάσεις αποδεκτών (containers) όπως είναι τα διανύσματα (vectors), οι αλγόριθμοι, και οι επαναλήπτες (iterators). Περισσότερα για αυτή τη κατηγορία στο κεφάλαιο **??**.
4. Input/ Output Stream Library: βιβλιοθήκες για την διαχείριση των ροών (π.χ. διαχείριση αρχείων).

Μια απλή συνάρτηση

Πέρα από τις καθιερωμένες συναρτήσεις, η C++ μας επιτρέπει να ορίσουμε και να χρησιμοποιούμε δικές μας συναρτήσεις.

Το παρακάτω πρόγραμμα τυπώνει μια γραμμή με δέκα παύλες '-' με τη χρήση συνάρτησης:

Πρόγραμμα 2.6: Ένα παράδειγμα για τη χρήση συνάρτησης

```
1  // simplefunction.cpp
2  // demonstrates simple function declaration, definition and calls
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  void line();   //function declaration
8
9  int main() {
10    int grade1 = 10, grade2 = 4, grade3 = 7;
11    line();  //call to function
12    cout  << setw (4) << "Name" << setw (6) << "Grade" <<endl;
13
14    line();  //call to function
15     cout << setw (4) << "John" << setw (6) << grade1 <<endl
16       << setw (4) << "Ross" << setw (6) << grade2 <<endl
17       << setw (4) << "Jane" << setw (6) << grade3 <<endl;
18     line(); //call to function
19
20    return 0;
21 }
22
23 // starline prints 10 stars '*' on a line.
24 // function definition
25 void line() {
26    for(int j=0; j<10; j++) //function body
27       cout << "-";
28    cout << endl;
29 }
```

Καλούμε μια δική μας συνάρτηση (Πρόγραμμα 2.6, γραμμή 11, 14, 18), αφού προηγουμένως την έχουμε δηλώσει (γραμμή 7) και υλοποιήσει (γραμμή 25-29).

Για συντομία, μπορούμε να παραλείψουμε τη δήλωση της συνάρτησης, αν την υλοποιούμε πριν από τη κλήση της. Για καλύτερη κατανόηση του κώδικα συνήθως επιλέγουμε να οργανώνουμε το πρόγραμμά μας (όσο αφορά τις συναρτήσεις) παραθέτοντας πρώτα τις δηλώσεις των συναρτήσεων, έπειτα τη `main()` συνάρτηση, και τέλος τις υλοποιήσεις των δικών μας συναρτήσεων.

Δήλωση συνάρτησης
Η δήλωση μιας συνάρτησης (function declaration, prototype) αποτελείται από τρία τμήματα με την εξής σειρά: (α) τον τύπο της τιμής που επιστρέφει, (β) το όνομα της συνάρτησης και (γ) τις παραμέτρους που δέχεται. Στο τέλος τοποθετούμε ελληνικό ερωτηματικό (;).

Στο πρόγραμμα 2.6 η πρόταση `void line();` δηλώνει τη συνάρτηση `line` που δεν επιστρέφει κάποια τιμή (`void`), και δεν δέχεται κάποιες παραμέτρους `()`.

Ορισμός συνάρτησης
Κάθε συνάρτηση που καλείται πρέπει προηγουμένως κάπου να έχει οριστεί. Ο ορισμός της συνάρτησης (function definition) αποτελείται από τη γραμμή της δήλωσής της ακολουθούμενη από τον κώδικα της συνάρτησης τοποθετημένο ανάμεσα σε αγκύλες .

Κλήσεις της συνάρτησης
Η κλήση σε μια συνάρτηση (call, invoke, execute a function) αποτελείται από το όνομα τη συνάρτησης ακολουθούμενο από τα τυπικά της ορίσματα μέσα σε παρενθέσεις. Στο πρόγραμμα 2.6 η συνάρτηση `line` δεν έχει ορίσματα `()`. Στο τέλος, όπως σε κάθε δήλωση, τοποθετούμε ελληνικό ερωτηματικό (;), π.χ. `line();`

Το πρόγραμμα 2.6 έχει σαν έξοδο:

```
----------
Name Grade
----------
John    10
Ross     4
Jane     7
----------
```

Επιστρεφόμενη τιμή

Μια συνάρτηση αφού εκτελεστεί ο κώδικας που περιέχει μπορεί στο τέλος να επιστρέψει μια τιμή. Συνήθως για τον υπολογισμό αυτής της τιμής χρησιμοποιούμε τη συνάρτηση.

Δείτε πως το πρόγραμμα 1.2 προσαρμόστηκε ώστε η μετατροπή δολαρίου σε ευρώ να λαμβάνει χώρα σε μια συνάρτηση η οποία επιστρέφει το αποτέλεσμα.

Πρόγραμμα 2.7: Μετατροπέας δολαρίων σε ευρώ με χρήση συνάρτησης

```
1  // currencywithfunction.cpp
2  // demonstrates functions that return a value
3  #include <iostream>
4  using namespace std;
5
6  const float EU = 0.73078;  // 1 us dollar is 0.73078 euro
7
8  double toEuro(double value);
9
```

```
10 int main() {
11
12    double usvalue;   // value in us dollars
13    cout << "Enter value in us dollars: ";
14    cin >> usvalue;
15    double euvalue = toEuro(usvalue); // value in euro
16    cout << "Equivalent value in euro: " << euvalue << endl;
17    return 0;
18 }
19
20 // returns the value in euro currency
21 double toEuro(double value){
22    return (value * EU); // value in euro
23 }
```

Για την επιστροφή τιμής χρησιμοποιούμε την εντολή `return` η οποία ακολουθείται από την τιμή που θέλουμε να επιστρέψει. Μια συνάρτηση μπορεί να επιστρέφει απλούς τύπους δεδομένων `char`, `short`, `int`, `float`, `double`, καθώς και τύπο δομής ή κλάσης (βλ. κεφάλαιο 3). Δεν μπορεί όμως άμεσα να επιστρέψει έναν τύπο πίνακα. Μπορεί όμως να επιστρέφει δείκτες σε οποιοδήποτε δομημένο στοιχείο (π.χ. δείκτη σε πίνακα). Όποιος και είναι ο τύπος που επιστρέφει θα πρέπει να συμφωνεί με τον τύπο που περιγράφει τόσο ο ορισμός όσο και η δήλωση της συνάρτησης.

Παραμετρικές Συναρτήσεις

Για να περάσουμε δεδομένα στις συναρτήσεις χρησιμοποιούμε τις παραμέτρους. Οι παράμετροι επιτρέπουν στις συναρτήσεις να λειτουργούν και να παράγουν διαφορετικό αποτέλεσμα κάτω από διαφορετικά δεδομένα.

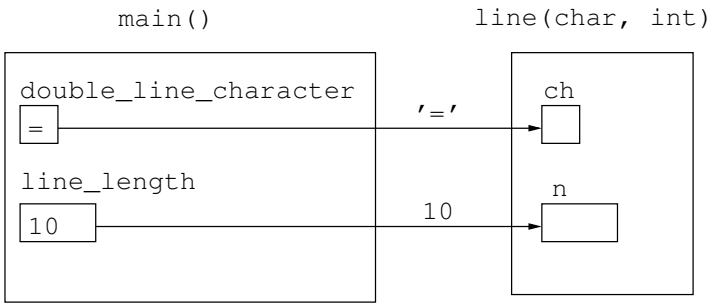
Η συνάρτηση στο πρόγραμμα 2.6 σε κάθε περίπτωση θα τυπώνει 10 παύλες. Παρακάτω, με τη χρήση παραμέτρων η συνάρτηση `line` τυπώνει τον `'ch'` χαρακτήρα `'n'` φορές:

```
1 void line(char ch, int n) {
2    for(int j = 0; j < n; j++) // function body
3       cout << ch;
4    cout << endl;
5 }
```

Ορισμός συνάρτησης με παραμέτρους
Ο ορισμός μιας συνάρτησης με παραμέτρους περιλαμβάνει τους τύπους των παραμέτρων, με ή χωρίς ονόματα (προτιμάμε τη χρήση ονομάτων μιας και δίνουν μια ιδέα για τη σημασία και τον τρόπο χρήσης τους). Οι παράμετροι μπορεί να είναι οποιουδήποτε, γνωστού στο πρόγραμμά μας, τύπου: ακέραιος, δεκαδικός, χαρακτήρας, συμβολοσειρά, πίνακας, διεύθυνση, δομή, κλάση...

```
1 void line(char ch, int n); // or void line(char, int);
```

Υπάρχουν δύο έννοιες που σχετίζονται με την κλήση συναρτήσεων: (α) η κλήση κατά τιμή (call by value), και (β) η κλήση κατά αναφορά (call by reference).



Σχήμα 2.1: Κλήση συνάρτησης με τιμή

Κλήση κατά τιμή

Για να περάσουμε τιμές στις παραμέτρους των συναρτήσεων χρησιμοποιούμε ορίσματα (arguments). Τα ορίσματα μπορεί να είναι είτε σταθερές τιμές (constants), είτε μεταβλητές:

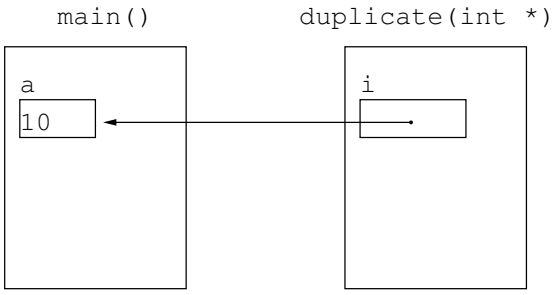
```
1  line('*', 30); // passing constants
2
3  char character = '*';
4  int num = 30;
5  line(character, num);   // passing variables
```

Μέσα στην συνάρτηση οι παράμετροι ορίζουν κανονικές μεταβλητές. Με τη κλήση κατά τιμή, οι παράμετροι αρχικοποιούνται στις τιμές που έχουμε περάσει ως ορίσματα (Σχήμα 2.1). Η εμβέλεια των παραμέτρων περιορίζεται στα πλαίσια της συνάρτησης (τοπικές μεταβλητές). Στην κλήση κατά τιμή οποιεσδήποτε αλλαγές λαμβάνουν χώρα στις τιμές των παραμέτρων δεν επηρεάζουν τις τιμές ορισμάτων.

Συνεπώς, με τη κλήση κατά τιμή προστατεύουμε τη τιμή των μεταβλητών που περνάμε ως ορίσματα από ανεπιθύμητες αλλαγές. Κάτι τέτοιο όμως μερικές φορές μας κοστίζει σε μνήμη μιας και για κάθε όρισμα δημιουργείται μία αντίγραφη μεταβλητή.

Πρόγραμμα 2.8: Ένα παράδειγμα κλήσης συνάρτησης κατά τιμή

```
1  // callbyvalue.cpp
2  // demonstrates simple function call by value
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  void line(char ch, int n);
8
9  int main() {
10    int grade1 = 10, grade2 = 4, grade3 = 7;
11    char double_line_char = '=';
12    char single_line_char = '-';
13    int line_length = 10;
14
15    line(double_line_char, line_length);   // function call by value
16    cout  << setw (4) << "Name" << setw (6) << "Grade" <<endl;
```



Σχήμα 2.2: Κλήση συνάρτησης κατά αναφορά

```
17
18     line(single_line_char, line_length);   // function call by value
19      cout << setw (4) << "John" << setw (6) << grade1 <<endl
20        << setw (4) << "Ross" << setw (6) << grade2 <<endl
21        << setw (4) << "Jane" << setw (6) << grade3 <<endl;
22
23      line(double_line_char, line_length);  // function call by value
24
25     return 0;
26  }
27
28  // prints 'ch' character 'n' times.
29  void line(char ch, int n) {
30     for(int j=0; j<n; j++)
31        cout << ch;
32     cout << endl;
33  }
```

Το πρόγραμμα 2.8 έχει σαν έξοδο:

```
==========
Name Grade
----------
John    10
Ross     4
Jane     7
==========
```

Κλήση κατά αναφορά

Στην κλήση κατά αναφορά (call by reference) στην συνάρτηση περνάμε την πραγματική μεταβλητή, και όχι απλά την τιμή της, δίνοντας πρόσβαση στη συνάρτηση για μεταβολές (Σχήμα **??**). Ένας τρόπος για να το κάνουμε αυτό είναι χρησιμοποιώντας δείκτες για παραμέτρους στις συναρτήσεις και διευθύνσεις στα ορίσματα των κλήσεών τους.

Το παρακάτω πρόγραμμα 2.9 καλεί κατά αναφορά μια συνάρτηση η οποία διπλασιάζει τη τιμή της παραμέτρου.

Πρόγραμμα 2.9: Ένα παράδειγμα κλήσης συνάρτησης κατά αναφορά

```
1  // callbyreference.cpp
2  // demonstrates simple function call by reference
3  #include <iostream>
4  using namespace std;
5
6  void dublicate(int *i);
7
8  int main() {
9
10    int a = 10;
11    dublicate(&a); // function call by reference
12    cout << a << endl;
13
14    return 0;
15 }
16
17 // dublicates i
18 void dublicate(int *i) {
19    (*i) *= 2;
20 }
```

Το πρόγραμμα 2.9 τυπώνει:

```
20
```

Με τη κλήση κατά αναφορά εξοικονομούμε μνήμη μιας και δεν δημιουργούνται αντίγραφα των παραμέτρων κάθε που καλούμε μια συνάρτηση. Παράλληλα, μας δίνεται η δυνατότητα με τη χρήση μιας συνάρτησης να περάσουμε πίσω στο πρόγραμμά μας περισσότερες από μία τιμές. Από την άλλη, τα ορίσματα μένουν απροστάτευτα σε τυχών ανεπιθύτες αλλαγές.

Χρήση `const`
Σε μερικές περιπτώσεις μπορεί να είναι χρήσιμο να έχουμε τη δυνατότητα να εκμεταλλευτούμε το γεγονός ότι η κλήση κατά αναφορά εξοικονομεί μνήμη και το γεγονός ότι η κλήση κατά τιμή προστατεύει τις μεταβλητές που περνάμε σαν παραμέτρους, ταυτόχρονα. Αυτό το πετυχαίνουμε με τη χρήση που προσδιοριστικού `const` καθώς δηλώνουμε τις παραμέτρους μιας συνάρτησης.

Στο παρακάτω παράδειγμα η συνάρτηση έχει πρόσβαση στην `i` μεταβλητή, αλλά μόνο για να την διαβάζει. Κάθε προσπάθεια αλλαγής της θα καταλήγει σε λάθος στον μεταγλωττιστή.

```
1  // return a duplicated i
2  int duplicate (const int *i) {
3     // (*i) *= 2;  // error: assignment of read-only location '* i'
4     return (*i) * 2;
5  }
```

Πολυμορφισμός

Υπάρχουν περιπτώσεις που θέλουμε η ίδια συνάρτηση να κάνει ακριβώς τις ίδιες λειτουργίες για διαφορετικούς όμως τύπους δεδομένων. Η C++ μας δίνει τη δυνατότητα υπερφόρτωσης μιας συνάρτησης (function over-

load), δηλαδή να ορίζουμε και να υλοποιούμε συναρτήσεις με το ίδιο όνομα αλλά με διαφορετικά ορίσματα (τύπους ή /και πλήθος) .

Για παράδειγμα, θέλουμε να ορίσουμε μια συνάρτηση η οποία θα διπλασιάζει μια τιμή είτε αυτή είναι ακέραια (`int`) είτε πραγματική (`double`).

Πρόγραμμα 2.10: Ένα παράδειγμα υπερφόρτωσης συνάρτησης

```
1  // functionoverload.cpp
2  // demonstrates function overloading
3  #include <iostream>
4  using namespace std;
5
6  int dublicate(int n);
7  double dublicate(double n);
8
9  int main() {
10
11    int a = 10;
12    cout << dublicate(a) << endl;
13
14    double b = 12.4;
15    cout << dublicate(b) << endl;
16    return 0;
17 }
18
19 // returns the dublicated integer n
20 int dublicate(int n) {
21    return ( n * 2);
22 }
23
24 // returns the dublicated double n
25 double dublicate(double n) {
26    return (n * 2);
27 }
```

Το πρόγραμμα 2.10 τυπώνει:

```
20
24.8
```

Ο μεταγλωττιστής επιλέγει ποια συνάρτηση θα εκτελέσει με βάση το όνομα και τους τύπους (και τη σειρά) των ορισμάτων

Η υπερφόρτωση είναι μια ιδέα πολυμορφισμού. Ο πολυμορφισμός, χαρακτηριστικό του αντικειμενοστραφή προγραμματισμού, επιτρέπει το ίδιο όνομα να χρησιμοποιείται για τεχνικής φύσεως διαφορετικούς σκοπούς Ο πολυμορφισμός μπορεί επίσης να εφαρμοστεί και στους τελεστές (Κεφάλαιο 4).

Ασκήσεις

2.1 Ο χρόνος μπορεί να περιγραφεί με τη χρήση τριών ακέραιων τιμών: δευτερόλεπτα (seconds), λεπτά (minutes) και ώρα (hour). Τα δευτερόλεπτα και τα λεπτά μπορούν να πάρουν τιμές από 0 έως 59, ενώ η ώρα από 0 έως 23.

(α) Σε C++, ορίστε μια δομή με όνομα `Time` που να αναπαριστά το χρόνο. Επειτα ορίστε 2 μεταβλητές τύπου `Time`.

(β) Δημιουργήστε μια συνάρτηση η οποία να ζητά από το χρήστη τιμές για τα δεδομένα μεταβλητών τύπου Time:
`void init(Time t);`
Δώστε τιμές στις δύο μεταβλητές.

(γ) Δημιουργήστε μια συνάρτηση η οποία να τυπώνει τα δεδομένα μιας μεταβλητής τύπου Time με τη μορφή `ωω:λλ:δδ`.
`void print(Time t);`
Τυπώστε τις δύο μεταβλητές.

(δ) Δημιουργήστε μια συνάρτηση η οποία να υπολογίζει και να επιστρέφει το συνολικό χρόνο σε δευτερόλεπτα.
`int toSeconds(Time t);`
Τυπώστε για κάθε μεταβλητή το χρόνο της σε δευτερόλεπτα.

(ε) Δημιουργήστε μια συνάρτηση που να ελέγχει την εγκυρότητα των τιμών ενός χρόνου και να μηδενίζει τιμές που βρίσκονται εκτός ορίων ενημερώνοντας κατάλληλα τον χρήστη.
`void validate(Time *t);`

2.2 Ένα σημείο (point) στον δισδιάστατο χώρο μπορεί να αναπαρασταθεί με τη χρήση δύο αριθμών: x, η τιμή του στον άξονα x, και y, η τιμή του στον άξονα y. Π.χ. (3, 2). Το άθροισμα δύο σημείων μπορεί να οριστεί ως το άθροισμα των τιμών για τον κάθε άξονα.

(α) Σε C++ ορίστε μια δομή που να αναπαριστά σημεία. Έπειτα ορίστε δύο σημεία και ζητήστε από το χρήστη να ορίσει τις τιμές για αυτά. Ορίστε ένα τρίτο σημείο ίσο με το άθροισμα των προηγούμενων δύο και τυπώστε το.

(β) Η απόσταση (d) ενός σημείου από την αρχή των αξόνων (0, 0) είναι όσο η τετραγωνική ρίζα του αθροίσματος των τετραγώνων κάθε άξονα:

$$d = \sqrt{x^2 + y^2}$$

Δημιουργήστε μια συνάρτηση η οποία να υπολογίζει και να επιστρέφει την απόσταση ενός σημείου από την αρχή των αξόνων (χρησιμοποιείστε τη συνάρτηση `sqrt` και `pow` την `cmath` βιβλιοθήκης.)

(γ) Ορίστε ένα πίνακα από σημεία, το μέγεθος του οποίου θα ζητάτε από το χρήστη να το ορίσει. Ζητήστε από το χρήστη να δώσει τιμές x, y για κάθε στοιχείο. Δημιουργήστε μια συνάρτηση η οποία θα ταξινομεί τα σημεία αυτά ανάλογα με την απόσταση που έχουν από την αρχή των αξόνων (π.χ. bubble sort).

2.3 Υλοποιείστε σε C++ ένα απλό παιχνίδι λέξεων. Ο πρώτος παίχτης εισάγει μια κρυφή λέξη, την οποία προσπαθεί να μαντέψει ο δεύτερος παίχτης. Ο δεύτερος παίχτης μπορεί να γνωρίζει τον αριθμό των χαρακτήρων της κρυφής λέξης, και τα σωστά γράμματα που μπορεί να περιέχει κάθε του προσπάθεια. Επιπλέον, μπορειτε να ζητήσετε από τον πρώτο παίχτη να δώσει μια μικρή βοήθεια.

```
Player1, give a secret word:
apple
```

```
Player1, give a hint:
fruit


Player2, venture to guess word of 5 characters
Hint: fruit
apricot
Player2, choose a word with 5 characters
Try again (y/n)?
y
grape
Correct letters: ----e
Try again (y/n)?
y
apple
Player2, correct guess!!!
The secret word was: apple
```

Χρησιμοποιείστε τις `strlen` και `strcmp` συναρτήσεις της `cstring` βιβλιοθήκης.

Προαιρετικά, σκεφτείτε τί μπορείτε να κάνετε για να αποκρίψετε από τον δεύτερο παίχτη την κρυφή λέξη.

2.4 Γράψτε σε C++ ένα πρόγραμμα που να εξομοιώνει το παιχνίδι της τράπουλας ``παπάς''. Σύμφωνα με τους κανόνες του παιχνιδιού, ο παπατζής (ο έμπειρος χαρτοπαίχτης) αρχικά δείχνει στον παίχτη τρία χαρτιά - συνήθως το ρήγας τριφύλλι (ο παπάς) και δύο αριθμούς. Στη συνέχεια αναποδογυρίζει τα χαρτιά και αλλάζει αρκετές φορές τις θέσεις τους μπροστά στα μάτια του παίχτη. Στο τέλος, ο παίκτης καλείται να απαντήσει που βρίσκεται ο παπάς. Αν μαντέψει σωστά, κερδίζει.

Χρησιμοποιείστε τη δομή `card` για να απεικονίσετε χαρτιά της τράπουλας:

```
1  struct card
2  {
3     int number; //2 to 10, jack, queen, king, ace
4     int suit;   //clubs, diamonds, hearts, spades
5  };
```

Το στοιχείο της δομής `number` παίρνει τιμές [2,14], με τους αριθμούς 11, 12, 13, 14 να αντιπροσωπεύουν τα Βαλές, Ντάμα, Ρήγας, Άσσος αντίστοιχα. Τα τέσσερα χρώματα της τράπουλας (`suit`) έχουν τιμή 0, 1, 2, ή 3, αντιπροσωπεύοντας αντίστοιχα τα Μπαστούνια, Καρό, Κούπες και Σπαθιά.

Μπορείτε να ορίσετε τις σταθερές:

```
1  const int clubs = 0; // suits
2  const int diamonds = 1;
3  const int hearts = 2;
4  const int spades = 3;
5
6  const int jack = 11; // face cards
7  const int queen = 12;
8  const int king = 13;
```

```
9  const int ace = 14;
```

ώστε να ορίζεται ένα χαρτί ως εξής:

```
1  card card1 = {7, clubs};
```

Στο πρόγραμμα, ορίστε 3 χαρτιά και ενημερώστε το χρήστη για την αρχική κατάσταση. Αλλάξτε τη θέση 2 χαρτιών μεταξύ τους και ενημερώστε το χρήστη, για όσες φορές θέλετε. Ζητήστε από το χρήστη να σας πει σε ποια θέση είναι ο παπάς.

(προαιρετικά, για να γίνει πιο δύσκολο το παιχνίδι, μπορείτε τα μηνύματα να τυπώνονται για ορισμένο χρόνο).

2.5 Αναθεωρείστε το παραπάνω πρόγραμμα χρησιμοποιώντας απαριθμήσεις.

2.6 Αναθεωρείστε το παραπάνω πρόγραμμα χρησιμοποιώντας συνάρτηση η οποία θα αναλαμβάνει την εναλλαγή των θέσεων 2 χαρτιών. Χρησιμοποιείστε την κλήση κατά αναφορά.

2.7 Σε C++, γράψτε τρεις συναρτήσεις που να υπολογίζουν και να επιστρέφουν τον μέγιστο αριθμό μεταξύ δύο, τριών, και τεσσάρων αριθμών (πολυμορφισμός).

Γράψτε πρόγραμμα που να ελέγχει τις συναρτήσεις αυτές.

Βοήθεια: για πιο σύντομο κώδικα, μπορείτε να ορίσετε την δεύτερη χρησιμοποιώντας την πρώτη, κ.ο.κ.

2.8 Σε C++, γράψτε μια συνάρτηση που να υπολογίζει το μέγιστο από ένα σύνολο ακεραίων. Γράψτε πρόγραμμα που να ελέγχει τη συνάρτηση αυτή.

2.9 Σε C++ γράψτε πρόγραμμα που να ελέγχει το αν μια ακολουθία χαρακτήρων που εισήγαγε ο χρήστης αποτελείται από γράμματα ή/και αριθμούς.

Βοήθεια: μπορείτε να χρησιμοποιείσετε την numeric `isalnum` συνάρτηση της `cctype` βιβλιοθήκης.

2.10 Αναζητήστε για την `string` βιβλιοθήκη.

Σε C++, αφού εισάγετε την `string` βιβλιοθήκη, ορίστε μια μεταβλητή τύπου `string` και δώστε της μια αρχική τιμή. Π.χ.:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main () {
6     string str = "the road not taken";
7     ...
8     return 0;
9  }
```

Τυπώστε την `str`.
Τυπώστε τη θέση που βρίσκεται για πρώτη φορά ο χαρακτήρας `t`.
Τυπώστε το πόσες φορές υπάρχει ο χαρακτήρας `t`.
Τυπώστε το μέγεθος της `str`.

Τυπώστε τους πέντε πρώτους χαρακτήρες της `str`.
Προσθέστε (append) μια συμβολοσειρά στο τέλος.
Τυπώστε τη νέα `str`.

Κεφάλαιο 3

Κλάσεις

κλάσεις - δεδομενα μέλη - συναρτήσεις μέλη - κατασκευαστές - συναρτήσεις κατάργησης - κατασκευαστές αντιγραφής - `const` συναρτήσεις μελη - αντικείμενα ως ορίσματα συναρτήσεων μελών - συναρτήσεις μέλη που επιστρέφουν αντικείμενα - πίνακες αριθμών και χαρακτήρων - πίνακες αντικειμένων - κλάσεις ως δεδομένα μέλη άλλων κλάσεων - δείκτες σε κλάσεις

Κλάσεις

Μία κλάση (class) είναι ένας καθορισμένος από το χρήστη τύπος δεδομένων. Αποτελεί μια συλλογή δεδομένων οποιουδήποτε τύπου (ιδιότητες), και συναρτήσεων (συμπεριφορές).

Ενα απλό παράδειγμα με κλάσεις

Ο παρακάτω κώδικας 3.1 ορίζει μια κλάση (`FooBar`), που περιέχει ένα ακέραιο δεδομένο (`data`), και δύο συναρτήσεις μέλη (`setData` και `getData`); .

Πρόγραμμα 3.1: Ένα απλό παράδειγμα με κλάσεις

```
1  // simpleclass.cpp
2  // demonstrates a simple class declaration and implementation
3  // in main() we create an object of this class and call its functions
4  #include <iostream>
5
6  using namespace std;
7
8  class FooBar { // declare a class
9     private:
10       int data; // member data
11    public:
12       void setData(int d); // member functions
13       int getData();
```

```
14 }; //end class
15
16 //implement member functions
17 void FooBar::setData(int d) {
18    data = d;
19 }
20
21 int FooBar::getData() {
22    return data;
23 }
24
25 int main() {
26    FooBar f; // define an object of type FooBar
27    f.setData(10); // call member function
28    cout << f.getData() << endl; //call member function
29
30    return 0;
31 }
```

Δήλωση κλάσης
Δηλώνουμε μια κλάση χρησιμοποιώντας το διευκρινιστικό `class` ακολουθούμενο από το όνομα της κλάσης (Πρόγραμμα 3.1 γραμμή 8-14). Το σώμα της κλάσης περικλείεται από αγκύλες . Στο τέλος τοποθετούμε ελληνικό ερωτηματικό (;).

Προσδιοριστές πρόσβασης
Στο σώμα της κλάσης μπορούμε να χρησιμοποιούμε προσδιοριστές πρόσβασης (access specifiers) οι οποίοι διευκρινίζουν τα δικαιώματα πρόσβασης στα δεδομένα και στις συναρτήσεις της κλάσης:

- private (ιδιωτική): τα μέλη είναι προσβάσιμα μόνο μέσα από την ίδια τη κλάση ή από φίλες συναρτήσεις (friend functions 4).
- protected (προστατευμένη): τα μέλη είναι προσβάσιμα μόνο μέσα από την ίδια τη κλάση ή φίλες συναρτήσεις ή παράγωγων (derived) κλάσεων (βλ. Κεφάλαιο 5).
- public (δημόσια): τα μέλη είναι προσβάσιμα και από έξω από την κλάση.

Η απόκρυψη ιδιοτήτων και συμπεριφορών μιας κλάσης από άλλες είναι χαρακτηριστικό του αντικειμενοστραφούς προγραμματισμού που ονομάζεται ενθυλάκωση (encapsulation ή data hinding). Αποκρύπτουμε στοιχεία της κλάσης μας για να τα προστατεύουμε από ανεπιθύμητες αλλαγές.

Δήλωση δεδομένων μελών
Μια κλάση μπορεί να περιέχει όσα δεδομένα μέλη (data members) θέλουμε. Δηλώνουμε τον τύπο και το όνομά τους κάτω από τον προσδιοριστή πρόσβασης που επιθυμούμε. Π.χ στο Πρόγραμμα 3.1 γραμμή 10 η κλάση έχει ένα ακέραιο ιδιωτικό μέλος, το `data`.

Δήλωση συναρτήσεων μελών
Οι συναρτήσεις μέλη (member functions) είναι συναρτήσεις που ανήκουν στη κλάση. Δηλώνουμε τις συναρτήσεις μέλη κάτω από τον προσδιοριστή πρόσβασης που επιθυμούμε. π.χ. στο Πρόγραμμα 3.1 στις γραμμές 12, 13 ορίζουμε 2 δημόσιες συναρτήσεις μέλη.

Ορισμός συναρτήσεων μελών
Μπορούμε να ορίζουμε τις συναρτησεις μέλη είτε μέσα είτε έξω από την κλάση. Όταν τις ορίζουμε έξω

από τη κλάση φροντίζουμε να προσδιορίζουμε πως οι συναρτήσεις ανήκουν στην κλάση δηλώνοντας το όνομά της και τον τελεστή `::` (scope resolution operator) πριν το όνομα της συνάρτησης (Πρόγραμμα 3.1 γραμμή 17-19, 21-23). Και στις δύο περιπτώσεις το σώμα της συνάρτησης περικλείεται από αγκύλες .

Ορισμός αντικειμένων
Η δήλωση μιας κλάσης δεν δημιουργεί αντικείμενα. Περιγράφει μόνο το πως θα μοιάζουν τα αντικείμενα. Ο ορισμός τους (definition, instantiation) δημιουργεί αντικείμενα. Στο Πρόγραμμα 3.1 γραμμή 26 ορίζεται το αντικείμενο `f`.

Κλήση συναρτήσεων μελών
Οι συναρτήσεις μέλη πάντα καλούνται μέσω ενός αντικειμένου της κλάσης στην οποία ανήκουν. Χρησιμοποιούμε τον τελεστή `.` ανάμεσα στο όνομα του αντικειμένου και στο όνομα της συνάρτησης μέλους (Πρόγραμμα 3.1 γραμμή 27, 28).

Κατασκευαστές

Οι κατασκευαστές (constructor) είναι συναρτήσεις ειδικού τύπου που μας επιτρέπουν να ορίζουμε αρχικές τιμές στα δεδομένα μέλη της κλάσης καθώς δημιουργούμε αντικείμενα. Οι κατασκευαστές έχουν το ίδιο όνομα με τη κλάση και δεν εχουν επιστρεφόμενη τιμή.

Πρόγραμμα 3.2: Ένα απλό παράδειγμα με κλάσεις και κατασκευαστές

```
1  // simpleclass2.cpp
2  // demonstrates constructors for a simple class
3  #include <iostream>
4
5  using namespace std;
6
7  class FooBar {
8     private:
9        int data; // member data
10    public:
11       FooBar(); // default constructor
12       FooBar(int d); // 1-arg constructor
13       void print(); // member functions
14 };
15
16 // implement constructors
17 FooBar::FooBar() {
18    data = 0;
19 }
20
21 FooBar::FooBar(int d) {
22    data = d;
23 }
24
25 // implement member functions
26 void FooBar::print() {
27    cout << data << endl;
28 }
```

```
29
30 int main() {
31    FooBar f1; // use first constructor
32    f1.print(); // prints 0
33
34    FooBar f2(40); // use 1-arg constructor
35    f2.print(); // prints 40
36
37    return 0;
38 }
```

Ο εξ ορισμού κατασκευαστής (default constructor) δεν περιέχει ορίσματα και μας επιτρέπει να αρχικοποιούμε τα δεδομένα μέλη με σταθερές τιμές (constant values). Π.χ. Πρόγραμμα 3.2 γραμμή 18.

Μπορούμε να ορίσουμε κατασκευαστές που περιέχουν παραμέτρους. Αυτοί μας επιτρέπουν να αρχικοποιούμε τα δεδομένα μέλη με τιμές που εχουν τα ορίσματα.

Ο μεταγλωττιστής κάθε φορά επιλέγει το κατασκευαστή που θα χρησιμοποιήσει σύμφωνα με τα ορίσματα που δηλώνουμε στον ορισμό ενός αντικειμένου.

Για να επιβεβαιώστε πως καλείτε ο κατασκευαστής ορίστε τον ως εξής:

```
1 FooBar::FooBar() {
2    cout << "constructor call" << endl;
3    data = 0;
4 }
```

Κατασκευαστής αντιγραφής

Ο κατασκευαστής αντιγραφής (default copy constructor) είναι ο κατασκευαστής που υπάρχει εξ ορισμού σε όλες τις κλάσεις. Μας επιτρέπει να αρχικοποιήσουμε ένα αντικείμενο μέσω ενός άλλου αντικειμένου του ίδιου τύπου.

Προσθέστε στη `main` του Προγράμματος 3.2

```
1 FooBar f3(f2); // default copy constructor call
2 cout << f3.getData() << endl; //prints 40
3
4 FooBar f4 = f2; // default copy constructor call
5 cout << f4.getData() << endl; //prints 40
```

Και στις δύο παραπανω περιπτώσεις καλείτε ο κατασκευαστής αντιγραφής, ο οποίος αναλαμβάνει να θέσει τις ίδιες τιμές των δεδομένων από το ένα αντικείμενο (`f2`) στο άλλο (`f3`, `f4`).

Συναρτήσεις Καταστροφής

Οι συναρτήσεις καταστροφής (destructos) ειναι ειδικού τύπου συναρτήσεις οι οποίες καλούνται για την καταστροφή, κατάργηση ενός αντικειμένου.

Οι συναρτήσεις καταστροφής έχουν το ίδιο όνομα με την κλάση, δεν έχουν καθόλου ορίσματα, και δεν επιστρέφουν κάποια τιμή.

Πρόγραμμα 3.3: Κλάσεις και συναρτήσεις καταστροφής

```
1  class FooBar {
2     private:
3        int data;
4     public:
5        FooBar() { // constructor
6           data = 0;
7     }
8        ~FooBar() { // destructor
9     }
10 };
```

Ενα ολοκληρωμένο παράδειγμα κλάσης

Η παρακάτω κλάση αναπαριστά αντικειμενα Απόστασης `Distance`. Κάθε απόσταση χαρακτηρίζεται από μέτρα (meters) και εκατοστά (centimeters). Για παράδειγμα 5m και 23cm.

Στη κλάση επίσης ορίζονται 2 κατακευστές.

Επιπλέον ορίζονται 4 συναρτήσεις μέλη:
η `void Distance::init();` που ζητά από το χρήστη τιμές για τα δεδομένα μέλη της απόστασης,
η `void Distance::print();` που τυπώνει τα δεδομένα μέλη της απόστασης,
η `void Distance::addCentimeters(int cm);` που προσθέτει `cm` εκατοστά στην απόσταση, και
η `int Distance::calculateCentimeters()` που επιστρεφει την απόσταση εκφρασμένη σε εκατοστά.

Πρόγραμμα 3.4: Ένα ολοκληρωμένο παράδειγμα με κλάσεις

```
1  // distance.cpp
2  // demonstrates a simple class
3  #include <iostream>
4
5  using namespace std;
6
7  class Distance {
8     private:
9        int meters;
10       int centimeters;
11    public:
12       Distance();
13       Distance(int m, int cm);
```

```
14        void init();
15        void print();
16        void addCentimeters(int cm);
17        int calculateCentimeters();
18  };
19
20  Distance::Distance() {
21     meters = 0;
22     centimeters = 0;
23  }
24
25  Distance::Distance(int m, int cm) {
26     meters = m;
27     centimeters = cm;
28  }
29
30  // get distance from user
31  void Distance::init() {
32     cout << "Enter meters: ";
33     cin >> meters;
34     cout << "Enter centimeters: ";
35     cin >> centimeters;
36  }
37
38  // print distance
39  void Distance::print() {
40     cout << meters << "meters, " << centimeters << "centimeters" <<endl;
41  }
42
43  // add 'cm' centimeters to distance
44  void Distance::addCentimeters(int cm) {
45     if ((cm + centimeters) >= 100) {
46        meters = meters + ((cm + centimeters)/100);
47        centimeters = (cm + centimeters)%100;
48     } else {
49        centimeters += cm;
50     }
51  }
52
53  // returns the distance in centimeters
54  int Distance::calculateCentimeters() {
55     return (meters*100 + centimeters);
56  }
57
58  int main() {
59     Distance d1;
60     cout << "initialize distance" << endl;
61     d1.init();
62     d1.print();
63
```

```
64     cout << "add 150cm to distance" << endl;
65     d1.addCentimeters(150);
66
67     cout << "distance became: ";
68     d1.print();
69
70     cout << "distance in centimeters: " << d1.calculateCentimeters() << endl;
71
72     return 0;
73 }
```

Το Πρόγραμμα 3.4 έχει σαν έξοδο:

```
initialize distance
Enter meters: 10
Enter centimeters: 55
10meters, 55centimeters
add 150cm to distance
distance became: 12meters, 5centimeters
distance in centimeters: 1205
```

Σταθερές Συναρτήσεις Μέλη

Το `const` διευκρινιστικό μετά την δήλωση μιας συνάρτησης υποδεικνύει ότι η συνάρτηση δεν θα τροποποιεί `private` μέλη της κλάσης (const member functions).

```
1  class Date {
2     int d ,m , y;
3     public:
4        int day() const {
5           return d;
6        }
7        void setDay(int days) const {
8           d = days; // error !!!
9        }
10 };
```

Σημειώστε πως υπάρχουν και σταθερές παράμετροι συναρτήσεων μελών (const member function arguments), καθώς και σταθερά αντικείμενα (const objects).

Στατικά δεδομένα μέλη

Τα στατικά (static) δεδομένα μέλη μιας κλάσης δημιουργούνται μία φορά, ανεξάρτητα του πόσα αντικείμενα της κλάσης δημιουργούνται.

Στο παρακάτω παράδειγμα η count κρατά τον αριθμό των αντικειμένων που δημιουργούνται.

Πρόγραμμα 3.5: Στατικά δεδομένα μέλη

```
1  // staticdata.cpp
2  // demonstrates static class data.
3  // FooBar keeps track of how many FooBar objects are
4  // created using the count static data member.
5  #include <iostream>
6
7  using namespace std;
8
9  class FooBar {
10    private:
11       static int count; //count declaration,
12                //only one count item for all objects
13    public:
14       FooBar() {
15          count++;
16       }
17       int getcount() const {
18          return count;
19       }
20 };
21
22 int FooBar::count = 0; //definition of count
23
24 int main() {
25    FooBar f1, f2, f3; //create three objects
26
27    cout << "count is " << f1.getcount() << endl; //each object
28    cout << "count is " << f2.getcount() << endl; //sees the
29    cout << "count is " << f3.getcount() << endl; //same value
30
31    return 0;
32 }
```

Το πρόγραμμα 3.5 τυπώνει:

```
count is 3
count is 3
count is 3
```

Αυτοαναφορά

Οι (μη στατικές) συναρτήσεις μέλη κάθε αντικειμένου έχουν πρόσβαση σε ένα δείκτη με το όνομα `this`, που δείχνει στο αντικείμενο για το οποίο κλήθηκε η συνάρτηση.

Πρόγραμμα 3.6: Ο δείκτης this

```cpp
// this.cpp
// demonstrates 'this' pointer
#include <iostream>

using namespace std;

class FooBar {
   private:
      int data;
   public:
      FooBar();
      FooBar(int d);
      void print() const;
      void printAddress() const;
      FooBar getGreater(FooBar f) const;
};

FooBar::FooBar() {
   this->data = 0; // access member data using this
}

FooBar::FooBar(int d) {
   this->data = d; // access member data using this
}

// prints the object's data using this
void FooBar::print() const {
   cout << "data: " << this->data << endl;
}

// prints the object's address using this
void FooBar::printAddress() const {
   cout << "Object's address: " << this << endl;
}

// returns the object having greater data using this
FooBar FooBar::getGreater(FooBar f) const {
   return f.data > data ? f : *this;
}

int main() {
   FooBar f1(2), f2(10);
   f1.print();
   f2.print();

   FooBar f3 = f1.getGreater(f2);
   f3.print();
   f3.printAddress();
}
```

Το πρόγραμμα 3.6 τυπώνει:

```
data: 2
data: 10
data: 10
Object's address: 0x7fff9bebdbf0
```

Χρησιμοποιούμε τον δείκτη `this` για να αποφύγουμε τη δημιουργία προσωρινών αντικειμένων. Περισσότερα παραδείγματα χρήσης στο Κεφάλαιο .

Χωρίζοντας μια κλάση σε περισσότερα αρχεία

Καθώς προσθέτουμε δεδομένα και συναρτήσεις μέλη σε κλάσεις, τα αρχεία τείνουν να γίνονται μεγάλα. Για την ευκολότερη κατανόηση και χρήση του κώδικα τον χωρίσουμε σε περισσότερα αρχεία.

Για παράδειγμα:

- Δηλώνουμε τη κλάση σε ένα αρχείο επικεφαλίδας `.h`, π.χ. `foobar.h`

```
1  // foobar.h
2  #include ...  // some header files
3
4  class FooBar {
5     private:
6        // private data members and functions
7        int data;
8        ...
9     public:
10       // constructors and public data members and functions
11       FooBar();
12       ...
13 };
```

- Ορίζουμε τους κατασκευαστές και τις συναρτήσεις μέλη της κλάσης σε ένα `.cpp` αρχείο, π.χ. `foobar.cpp`, στο οποίο ωφείλουμε να συμπεριλάβουμε το αντίστοιχο αρχείο επικεφαλίδας, π.χ. `#include ''foobar.h''`

```
1  // foobar.cpp
2  #include ...  // some header files
3  #include "foobar.h"
4
5  FooBar::FooBar() {
6     data = 0;
7  }
8
9  ... // the rest definitions
```

- Σε ένα τρίτο αρχείο `.cpp` συμπεριλαμβάνουμε το αρχείο επικεφαλίδας και ορίζουμε τη `main`, στην οποία μπορούμε να δημιουργούμε αντικείμενα της κλάσης.

```
1  //main.cpp
2  #include ...  // some header files
3  #include "foobar.h"
4
5  int main() {
6     FooBar f;
7     ...
8  }
```

Αντικείμενα ως ορίσματα συναρτήσεων μελών

Οι συναρτήσεις μέλη μιας κλάσης μπορούν να περιέχουν παραμέτρους μεταβλητές τύπου κλάσης.

Για παράδειγμα η συνάρτηση μέλος `addDistance` στο Πρόγραμμα 3.7 προσθέτει ένα αντικείμενο τύπου Distance στο τρέχον:

Πρόγραμμα 3.7: Αντικείμενα ως ορίσματα συναρτήσεων μελών

```
1  // distance2.cpp
2  // demonstrates classes and objects as function arguments
3
4  #include <iostream>
5
6  using namespace std;
7
8  class Distance {
9     private:
10       int meters;
11       int centimeters;
12    public:
13       Distance();
14       Distance(int m, int cm);
15       void print() const;
16       void addDistance(const Distance& dist);
17 };
18
19 Distance::Distance() {
20    meters = 0;
21    centimeters = 0;
22 }
23
24 Distance::Distance(int m, int cm) {
25    meters = m;
26    centimeters = cm;
27 }
```

```
28
29 // print distance
30 void Distance::print() const {
31    cout << meters << "meters, " << centimeters << "centimeters" <<endl;
32 }
33
34 void Distance::addDistance(const Distance& dist) {
35    meters += dist.meters;
36    centimeters += dist.centimeters;
37
38    //if sum exceeds 100, decrease centimeters by 100 and increase meters by 1
39    if (centimeters >= 100) {
40       meters += 1;
41       centimeters -= 100;
42    }
43 }
44
45 int main() {
46    Distance d1(2, 3), d2(4, 1);
47    cout<< "d1: ";
48    d1.print();
49
50    cout<< "d2: ";
51    d2.print();
52
53    cout << "Adding d2 to d1..." << endl;
54    d1.addDistance(d2);
55    cout << "d1: ";
56    d1.print(); //prints 6, 4
57
58    return 0;
59 }
```

Παρατηρήστε το γεγονός ότι η συνάρτηση μέλος μιας κλάσης έχει πρόσβαση στα δεδομένα μέλη της ίδιας της κλάσης καθώς και των αντικειμένων ιδίου τύπου που μπορεί να περνιούνται ως ορίσματα.

Το Πρόγραμμα 3.7 έχει σαν έξοδο:

```
d1: 2meters, 3centimeters
d2: 4meters, 1centimeters
Adding d2 to d1...
d1: 6meters, 4centimeters
```

Συναρτήσεις μέλη που επιστρέφουν αντικείμενα

Οι συναρτήσεις μέλη μιας κλάσης μπορούν να επιστρέφουν αντικείμενα.

Για παράδειγμα η συνάρτηση μέλος `addDistance` στο Πρόγραμμα 3.8 προσθέτει ένα αντικείμενο τύπου Distance (`d`) με το τρέχον και το αποτέλεσμα το αποθηκεύει σε ένα τρίτο αντικείμενο (`tmp`), το οποίο και

επιστρέφει:

Πρόγραμμα 3.8: Συναρτήσεις μέλη που επιστρέφουν αντικείμενα

```
1  // distance3.cpp
2  // demonstrates a member function that returns an object
3  #include <iostream>
4
5  using namespace std;
6
7  class Distance {
8     private:
9        int meters;
10       int centimeters;
11    public:
12       Distance();
13       Distance(int m, int cm);
14       void print() const;
15       Distance addDistance(const Distance& dist);
16 };
17
18 Distance::Distance() {
19    meters = 0;
20    centimeters = 0;
21 }
22
23 Distance::Distance(int m, int cm) {
24    meters = m;
25    centimeters = cm;
26 }
27
28 // print distance
29 void Distance::print() const {
30    cout << meters << "meters, " << centimeters << "centimeters" <<endl;
31 }
32
33 // add d with this distance and return it as a new object
34 Distance Distance::addDistance(const Distance& dist) {
35    Distance tmp;
36    // add meters
37    tmp.meters = meters + dist.meters;
38    tmp.centimeters = centimeters  + dist.centimeters;
39
40    //if sum exceeds 100, decrease centimeters by 100 and increase meters by 1
41    if (tmp.centimeters >= 100) {
42       tmp.meters += 1;
43       tmp.centimeters -= 100;
44    }
45
46    return tmp;
```

```
47 }
48
49 int main() {
50    Distance d1(3, 50), d2(4, 60), d3;
51    cout<< "d1: ";
52    d1.print();
53
54    cout<< "d2: ";
55    d2.print();
56
57    cout<< "d3: ";
58    d3.print();
59
60    cout << "Adding d1 and d2 to d3..." << endl;
61    d3 = d1.addDistance(d2);
62    cout<< "d3: ";
63    d3.print();
64
65    return 0;
66 }
```

Το Πρόγραμμα 3.8 έχει σαν έξοδο:

```
d1: 3meters, 50centimeters
d2: 4meters, 60centimeters
d3: 0meters, 0centimeters
Adding d1 and d2 to d3...
d3: 8meters, 10centimeters
```

Συμβολοσειρές και κλάσεις

Στη C++ υπάρχουν δύο κύριοι τρόποι για να ορίσουμε μια συμβολοσειρά: ο ένας είναι με τη χρήση πίνακα χαρακτήρων (c style) και ο άλλος με τη χρήση της κλάσης `string`.

Σύμφωνα με τον πρώτο τρόπο μια συμβολοσειρά είναι ένας πίνακας από χαρακτήρες:

```
1 char name[30]; // an array of 30 characters
```

Διαχειριζόμαστε μεταβλητές τέτοιου τύπου εύκολα με τη χρήση συναρτήσεων της `cstring` βιβλιοθήκης. Π.χ. για να αντιγράψουμε μια συμβολοσειρά σε μία άλλη χρησιμοποιούμε την `strcpy`.

```
char* strcpy (char *destination, const char *source);
```

Στο παρακάτω παράδειγμα ορίζεται μια κλάση `Person` η οποία αναπαριστά πρόσωπα, κρατώντας το όνομα, τη διεύθυνση, την ηλικία και το ύψος τους.

Παρατηρήστε τον τρόπο χρήσης των συμβολοσειρών. Σημειώστε πως, όπως και στη C, οι πίνακες ως ορίσματα συναρτήσεων περνιούνται πάντα κατά αναφορά.

Πρόγραμμα 3.9: Συμβολοσειρές ως δεδομένα μέλη κλάσεων

```
1  // person.cpp
2  // demonstrates the use of arrays as data members
3  #include <iostream>
4  #include <cstring>
5
6  using namespace std;
7
8  class Person {
9     private:
10       char name[20];
11       char address[50];
12       float age;
13       float height;
14    public:
15       Person();
16       Person(char n[], char ad[], float ag, float h);
17       void print() const;
18 };
19
20 Person::Person() {
21    strcpy(name, "");
22    strcpy(address, "");
23    age = 0;
24    height = 0;
25 }
26
27 Person::Person(char n[], char ad[], float ag, float h) {
28    strcpy(name, n);
29    strcpy(address, ad);
30    age = ag;
31    height = h;
32 }
33
34 void Person::print() const {
35    cout << "Name: " << name << ", address: " << address
36       << " age: " << age << " , height: " << height << endl;
37 }
38
39
40 int main() {
41    Person john ("John", "76th Stafforf Str", 22, 1.80);
42    john.print();
43
44    return 0;
45 }
```

Σύμφωνα με τον δεύτερο τρόπο, ορίζουμε αντικείμενα της κλάσης `string`:

```
#include <iostream>
\\ ...

void f() {
   string str = "This is a string.";
   cout << str; // prints the str
   \\...
}
```

Διαχειριζόμαστε μεταβλητές τέτοιου τύπου εύκολα με τη χρήση συναρτήσεων μελών της `string` κλάσης. Π.χ. για να αντιγράψουμε μια συμβολοσειρά σε μία άλλη χρησιμοποιούμε χρησιμοποιούμε τον υπερφορτωμένο τελεστή ανάθεσης =.

```
string str1 = "Another string";

string str2;
str2 = str1;  // str2 becomes "Another string."
```

Στο παρακάτω παράδειγμα ορίζεται η προηγουμενη κλάση `Person` με τη χρήση της `string`.

Πρόγραμμα 3.10: C++ συμβολοσειρές ως δεδομένα μέλη κλάσεων

```
// person_with_cpp_strings.cpp
// demonstrates the use of c++ string
#include <iostream>
#include <string> // its "string", not "cstring"

using namespace std;

class Person {
   private:
      string name;
      string address;
      float age;
      float height;
   public:
      Person();
      Person(const string& n, const string& ad, float ag, float h);
      void print() const;
};

Person::Person() {
   name = "";
   address = "";
   age = 0;
   height = 0;
}

```

```
27 Person::Person(onst string& n, const string& ad, float ag, float h) {
28    name = n;
29    address = ad;
30    age = ag;
31    height = h;
32 }
33
34 void Person::print() const {
35    cout << "Name: " << name << ", address: " << address
36       << " age: " << age << " , height: " << height << endl;
37 }
38
39
40 int main() {
41    Person john ("John", "76th Stafford Str", 22, 1.80);
42    john.print();
43
44    return 0;
45 }
```

Τα Πρόγραμμα 3.9 και 3.10 έχουν σαν έξοδο:

```
Name: John, address: 76th Stafford Str age: 22 , height: 1.8
```

Πίνακες από αντικείμενα

Μπορούμε να ορίσουμε πίνακες από αντικείμενα. Για παράδειγμα, ο παρακάτω πίνακας `distances` είναι πίνακας 10 αποστάσεων (distance):

```
1 Distance distances[10]; // an array of 10 distances
```

Αναφερόμαστε στα δημόσια δεδομένα μέλη και συναρτήσεις κάθε απόστασης χρησιμοποιώντας τον τελεστή τελεία (.), αφού πρώτα προσδιορίσουμε τη θέση του στον πίνακα, π.χ.

```
1 distances[1].setMeters(20); // meters of the 2nd distance becomes 20
```

Στο παρακάτω παράδειγμα ορίζουμε ένα πίνακα από 3 distances και τον τυπώνουμε.

Πρόγραμμα 3.11: Πίνακες από αντικείμενα

```
1 // distancearray.cpp
2 // demonstrates arrays of objects
3
4 #include <iostream>
5 using namespace std;
6
```

```
 7  class Distance {
 8     private:
 9        int meters;
10        int centimeters;
11     public:
12        Distance();
13        Distance(int m, int cm);
14        int getMeters() const;
15        void setMeters(int m);
16        int getCentimeters() const;
17        void setCentimeters(int cm);
18        void print() const;
19  };
20
21  Distance::Distance() {
22     meters = 0;
23     centimeters = 0;
24  }
25
26  Distance::Distance(int m, int cm) {
27     meters = m;
28     centimeters = cm;
29  }
30
31  int Distance::getMeters() const{
32     return meters;
33  }
34
35  void Distance::setMeters(int m) {
36     meters = m;
37  }
38
39  int Distance::getCentimeters() const {
40     return centimeters;
41  }
42
43  void Distance::setCentimeters(int cm) {
44     centimeters = cm;
45  }
46
47  void Distance::print() const {
48     cout << meters << "meters, " << centimeters << "centimeters" <<endl;
49  }
50
51  // prints all distances of an array of Distances
52  void printAllDistances(Distance dists[], int size);
53
54  const int SIZE = 3; // number of distances in array
55
56  int main() {
```

```
57    Distance dists[SIZE]; // array of distances
58
59    // get 10 distances
60    cout << "Give " <<  SIZE << " distances"<< endl;
61    for (int i=0; i<SIZE; i++) {
62       int tmp_meters = 0;
63       cout << "Give meters for distance " << i+1 << ": ";
64       cin >> tmp_meters;
65       dists[i].setMeters(tmp_meters);
66
67       int tmp_centimeters = 0;
68       cout << "Give centimeters for distance " << i+1 << ": ";
69       cin >> tmp_centimeters;
70       dists[i].setCentimeters(tmp_centimeters);
71    }
72    // print them back
73    cout << endl << "You gave:" << endl;
74    printAllDistances(dists, SIZE);
75 }
76
77 /**
78  * prints all distances of an array of Distances
79  * @param dists[] an array of Distances
80  * @param size the size of the array
81  */
82 void printAllDistances(Distance dists[], int size) {
83    for (int i=0; i<size; i++) {
84       dists[i].print();
85    }
86 }
```

```
Give 3 distances
Give meters for distance 1: 23
Give centimeters for distance 1: 65
Give meters for distance 2: 23
Give centimeters for distance 2: 6
Give meters for distance 3: 245
Give centimeters for distance 3: 35

You gave:
23meters, 65centimeters
23meters, 6centimeters
245meters, 35centimeters
```

Δείκτες σε αντικείμενα

Μπορούμε να ορίσουμε δείκτες σε αντικείμενα. Για παράδειγμα, ο παρακάτω δείκτης `ptr` είναι δείκτης σε αντικείμενο τύπου `Person`:

```
1  Person *ptr;
```

Ο `ptr` δείχνει στον `john`:

```
1  Person john ("John", "76th Stafford Str", 22, 1.80);
2  ptr = &john;   // ptr points to john
```

Για να αναφερθούμε σε δημόσια στοιχεία του αντικειμένου μέσω του δείκτη χρησιμοποιούμε τον τελεστή `->` (arrow operator), π.χ.:

```
1  ptr->setAge(25);  // john' s age becomes 25;
```

Στο παρακάτω παράδειγμα, χρησιμοποιούμε δείκτες σε αντικείμενα τύπου `Person` προκειμένου να μας επιστραφεί ο `Person` με τη μεγαλύτερη ηλικία (κλήση κατά αναφορά).

Πρόγραμμα 3.12: Δείκτες σε αντικείμενα και κλήση κατά αναφορά

```
1  // person2.cpp
2  // demonstrates pointers of objects
3  #include <iostream>
4  #include <string>
5
6  using namespace std;
7
8  class Person {
9     private:
10       string name;
11       string address;
12       float age;
13       float height;
14    public:
15       Person();
16       Person(const string& n, const string& ad, float ag, float h);
17       float getAge() const;
18       void setAge(float ag);
19       string getName() const;
20       void print() const;
21 };
22
23 Person::Person() {
24    name = "";
25    address = "";
```

```
26     age = 0;
27     height = 0;
28  }
29
30  Person::Person(const string& n, const string& ad, float ag, float h) {
31     name = n;
32     address = ad;
33     age = ag;
34     height = h;
35  }
36
37  float Person::getAge() const {
38     return age;
39  }
40
41  void Person::setAge(float ag) {
42     age = ag;
43  }
44
45  string Person::getName() const {
46     return name;
47  }
48
49  void Person::print() const {
50     cout << "Name: " << name << ", address: " << address
51        << " age: " << age << " , height: " << height << endl;
52  }
53
54  /* Compares the ages of two persons and return the older Person */
55  Person& older(Person *person1, Person *person2) {
56     if (person1->getAge() >  person2->getAge())
57        return *person1;
58     else
59        return *person2;
60  }
61
62  int main() {
63     Person john ("John", "76th Stafford Str", 22, 1.80);
64     john.print();
65     Person jane ("Jane", "34th Oxford Str", 21, 1.60);
66     jane.print();
67
68     Person olderperson = older(&john, &jane);
69     cout << "The older person is: " << olderperson.getName() << endl;
70
71     return 0;
72  }
```

```
Name: John, address: 76th Stafford Str age: 22 , height: 1.8
```

```
Name: Jane, address: 34th Oxford Str age: 21 , height: 1.6
The older person is: John
```

Κλάσεις ως δεδομένα μέλη άλλων κλάσεων

Τα δεδομένα μέλη μιας κλάσης μπορεί να είναι τύπου μιας άλλης κλάσης.

Το παρακάτω παράδειγμα ορίζει μια κλάση που αναπαριστά δωμάτια (`Room`). Κάθε δω μάτιο χαρακτηρίζεται από το μήκος και το πλάτος του. Το μήκος και το πλάτος είναι τύπου `Distance`.

Πρόγραμμα 3.13: Κλάσεις ως δεδομένα μέλη κλάσεων

```
1  // room.cpp
2  // demonstrates the use of classes as data members
3  #include <iostream>
4  #include "distance.h"
5
6  using namespace std;
7
8  class Room {
9     private:
10       Distance width;
11       Distance length;
12    public:
13       Room();
14       Room(const Distance& w, const Distance& l);
15       void print() const;
16       float getArea() const;
17 };
18
19 Room::Room(const Distance& w, const Distance& l) {
20    width = w; // default copy constructor
21    length = l;
22 }
23
24 void Room::print() const {
25    cout << "width: "; width.print();
26    cout << "length: "; length.print();
27 }
28
29 float Room::getArea() const {
30    return width.calculateCentimeters() * length.calculateCentimeters() / 10000;
31 }
32
33 int main() {
34    Room kitchen(Distance(3, 0), Distance(4, 0)); // define an object of type Room
35    kitchen.print();
36    cout << "kitchen's area: "<< kitchen.getArea()
37       << "m^2" << endl;
```

```
38     return 0;
39 }
```

Το πρόγραμμα 3.13 τυπώνει:

```
width: 3meters, 0centimeters
length: 4meters,0centimeters
kitchen's area: 12m^2
```

Δώστε προσοχή στην χρήση του default copy constructor καθώς και στον ορισμό των αντικειμένων τύπου `Room`.

Πίνακες ως δεδομένα μέλη κλάσεων

Τα δεδομένα μέλη μιας κλάσης μπορεί να είναι πίνακες από αντικείμενα. Στο παρακάτω παράδειγμα ένας μαθητής (`Student`) έχει όνομα, ενα πίνακα από δέκα μαθήματα, και αντίστοιχα ένα πίνακα για τις βαθμολογίες του σε αυτά.

```
 1 class Course {
 2    private:
 3       string title;
 4    // ...
 5 };
 6
 7 class Student {
 8    private:
 9       string name;
10       Course courses[10]; // an array of student's courses
11       float grades[10];   // an array of student's grades
12       // ...
13 };
```

Ασκήσεις

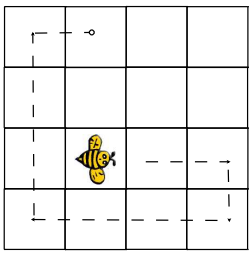
3.1 Δημιουργείστε μια κλάση με όνομα `Time` που έχει τρία δεδομένα μέλη: τις ώρες (hours), τα λεπτά (minutes) και τα δευτερόλεπτα (seconds). Ο ένας κατασκευαστής αρχικοποιεί τα δεδομένα σε μηδέν και ο άλλος σε συγκεκριμένες τιμές που περνιούνται ως ορίσματα. Μια συνάρτηση μέλος τυπώνει την ώρα με τη μορφή hh:mm:ss. Μία ακόμα συνάρτηση μέλος προσθέτει δύο αντικείμενα τύπου ώρας τα οποία περνιούνται ως ορίσματα.

Στη `main()` ορίστε δύο αντικείμενα τύπου `Time` χρησιμοποιώντας τον δεύτερο κατασκευαστή και ένα με τον πρώτο. Προσθέστε τα δύο πρώτα στο τρίτο αντικείμενο. Τυπώστε όλα τα αντικείμενα.

3.2 Σε C++ δημιουργείστε μια κλάση Employee που να έχει δύο δεδομένα μέλη: έναν ακέραιο για τον κωδικό του υπαλλήλου και έναν δεκαδικό για τον μισθό του. Υλοποιείστε μια συναρτήση μέλος που θα ζητά από το χρήστη τιμές για τα δεδομένα μέλη, και μία που να τυπώνει τα δεδομένα μέλη.

Στη `main()` ορίστε δύο αντικείμενα τύπου Employee και καλέστε τη συνάρτηση για να δώσει ο χρήστης δεδομένα και τυπώστε τους υπαλλήλους.

3.3 Θεωρείστε πως είμαστε σε ένα επίπεδο χώρο 4x4 στο οποίο μπορούν να υπάρχουν και να κινούνται διάφορα αντικείμενα, π.χ. ανθρωποι, αυτοκίνητα, ζώα.



Σε ένα τέτοιο χώρο, ένα αντικείμενο (όπως η μέλισσα του διπλανού σχήματος) έχει 3 ιδιότητες: την θέση του σε σχέση με τον αξονα x και y (ακέραιοι), και τον προσανατολισμό του. Για ευκολία, ορίστε την απαρίθμηση

```
enum direction {east, west, north, south};
```

Δίπλα η μέλισσα είναι στη (2, 2) θέση και κοιτά ανατολικά (east).

Επιπλέον, ένα αντικείμενο έχει τις εξής συμπεριφορές: (α) επιτρέπει να ορίσει κανείς την θέση του `void setPosition(int, int);`, (β) τυπώνει την θέση του `void showPosition();`, (γ) κινείται `void move()`, αυξάνοντας ή μειώνοντας το x ή το y κατά ένα, σύμφωνα με τον προσανατολισμό του (για παράδειγμα, αφού η μέλισσα βλέπει ανατολικά η επόμενή της θέση είναι το κουτί `x=3, y=2`) (δ) γυρίζει προς τα δεξιά (αλλάζει προσανατολισμό) `void turnRight()` (για παράδειγμα αν κοιτά ανατολικά, γυρνά νότια). (ε) ελέγχει το αν μπορεί να μετακινηθεί σε επόμενη θέση `bool canMove()`. Σκεφτείτε πως η νέα θέση θα πρέπει να είναι μέσα στα όρια του χώρου [1,4].

Υλοποιείστε σε C++ μια κλάση `animal` με τις παραπάνω ιδιότητες και συμπεριφορές. Στην `main` ορίστε ένα αντικείμενο και βάλτε το να αλλάζει 10 θέσεις. Οταν ενα αντικείμενο δεν μπορεί να κινηθεί να γυρίζει προς τα αριστερα, ώστε να συνεχίσει να κινείται σε άλλη κατεύθυνση. Για παράδειγμα, η μέλισσα δίπλα θα κάνει την παρακάτω διαδρομή:

```
Position x=2, y=2
Position x=3, y=2
Position x=4, y=2
Position x=4, y=1
Position x=3, y=1
Position x=2, y=1
Position x=1, y=1
Position x=1, y=2
Position x=1, y=3
Position x=1, y=4
Position x=2, y=4
```

3.4 Ένα κλάσμα αποτελείται από τον αριθμητή και τον παρονομαστή. Σε C++ ορίστε μία κλάση που να αναπαριστά κλάσματα. Ορίστε κατάλληλους κατασκευαστές και μια συνάρτηση μέλος που να τυπώνει το κλάσμα.

Ορίστε μια συνάρτηση μέλος η οποία να προσθέτει ένα κλάσμα με το τρέχον.

Τελος, ορίστε μια συνάρτηση μέλος που να απλοποιεί τον αριθμητή και παρονομαστή του κλάσματος, βρίσκοντας τον μέγιστο κοινό διαιρέτη.

Στη `main()` ορίστε δύο κλάσματα. Προσθέστε το δεύτερο κλάσμα στο πρώτο. Απλοποιείστε το δεύτερο κλάσμα.

3.5 Να υλοποιηθεί σε C++ μια κλάση η οποία να αναπαριστά ημερομηνίες (`Date`). Συμπεριλάβετε μια συνάρτηση μέλος η οποία να τυπώνει ημερομηνίες με τη μορφή:

dd days, *mm* months, *yyyy* years

Επιπλέον, υλοποιήστε συνάρτηση η οποία θα υπολογίζει την διαφορά δύο ημερομηνιών, και να την επιστρέφει εκφρασμένη σε `Date`. Θεωρείστε πως κάθε μήνας έχει 30 μέρες.

Στη `main()` δηλώστε δύο ημερομηνίες: την σημερινή (`today`) και την ημερομηνία γενεθλίων σας (`mybirthday`) και τυπώστε τες.

Στη συνέχεια υπολογίστε και τυπώστε την διαφορά τους.

Παράδειγμα εξόδου:

```
Today: 2 days, 12 months, 2008 years
My birthday: 9 days, 2 months, 1989 years
Time past since my birthday: 23 days, 9 months, 19 years
```

3.6 Να υλοποιηθεί σε C++ μια κλάση η οποία να αναπαριστά την ώρα (`Time`).

Συμπεριλάβετε στην κλάση μια συνάρτηση μέλος η οποία να τυπώνει την ώρα με τη μορφή:

ss secs, *mm* mins, *hh* hours

Επιπλέον, υλοποιήστε συνάρτηση μέλος η οποία θα υπολογίζει την διαφορά της ώρας με μία δεύτερη και να την επιστρέφει εκφρασμένη σε `Time`.

Στη `main()` δηλώστε δύο ώρες: μια αρχική (`start`) και μια τελική (`finished`) και τυπώστε τες.

Στη συνέχεια υπολογίστε και τυπώστε την διαφορά τους.

Παράδειγμα εξόδου:

```
Started: 12secs, 20mins, 13hours
Finished: 10secs, 10mins, 17hours
Time passed: 58secs, 49mins, 3hours
```

3.7 Να γράψετε σε C++ μια κλάση που να αναπαριστά Πίξελ (`Pixel`). Η κλάση θα πρέπει να κρατά τρεις ιδιωτικές (`private`), ακέραιες μεταβλητές που να δηλώνουν την ποσότητα κόκκινου (`red`), πράσινου (`green`) και μπλε (`blue`) που έχει ένα πίξελ.

Ορίστε κατασκευαστές και μια συνάρτηση μέλος που να τυπώνει τα στοιχεία ενός πιξελ με τη μορφή:

Επιπλέον, υλοποιήστε συνάρτηση μέλος `calculateColorAverage` η οποία να υπολογίζει και να επιστρέφει το μέσο όρο των χρωμάτων του πιξελ.

Υλοποιήστε συνάρτηση μέλος `isLighter`, η οποία συγκρίνει ένα πίξελ p με το τρέχον και επιστρέφει αληθές (`true`) όταν το τρέχον έχει μέσο όρο χρωμάτων μεγαλύτερο του p, και ψευδές (`false`) σε οποιαδήποτε άλλη περίπτωση.

Στη `main()` ορίστε δύο πίξελ. Συγκρίνετε την φωτεινότητά τους και τυπώστε το συμπέρασμα.

Αν τα πίξελ μιας εικόνας φυλάσσονται σε ένα μονοδιάστατο πίνακα από πίξελ, γράψτε μια (εξωτερική) συνάρτηση `calculateImageColorAverage` που να επιστρέφει το μέσο χρώμα της εικόνας.

Έπειτα στη `main()` ορίστε ένα πίνακα από πίξελ με μέγεθος 2 (`image[2]`) και αναθέστε σε αυτόν τα δύο πίξελ. Στη συνέχεια τυπώστε το μέσο χρώμα της εικόνας `image`.

3.8 Να γράψετε σε C++ μια κλάση που να αναπαριστά Σημεία (`Point`). Η κλάση θα πρέπει να κρατά δύο ιδιωτικές (private), ακέραιες μεταβλητές που να δηλώνουν την απόσταση του σημείου από τον άξονα `x` (x), και από τον άξονα `y` (y).

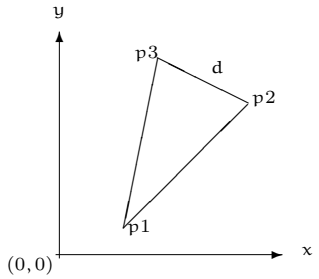
Υλοποιήστε κατασκευαστές και συνάρτηση μέλος που να τυπώνει τα στοιχεία ενός σημείου με τη μορφή:

```
(x , y)
```

Στη `main()` ορίστε τρία σημεία `p1`, `p2`, `p3` και τυπώστε τα.

Υλοποιήστε μια συνάρτηση-μέλος `double distance(Point p)` η οποία να υπολογίζει και επιστρέφει την απόσταση μεταξύ του τρέχοντος σημείου και του σημείου `p`. Η απόσταση μεταξύ δύο σημείων (x_1, y_1) και (x_2, y_2) υπολογίζεται από τον τύπο

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



Σημείωση: Κάντε χρήση των συναρτήσεων `sqrt` και `pow` της βιβλιοθήκης `cmath`.

Υλοποιήστε μια συνάρτηση-μέλος `bool hasGreaterDistance(Point p)` η οποία συγκρίνει την απόσταση του τρέχοντος σημείου από το σημείο (0, 0) και την απόσταση του `p` από το (0,0) και επιστρέφει αληθές (`true`) όταν η πρώτη (απόσταση) είναι μεγαλύτερη από τη δεύτερη και ψευδές σε οποιαδήποτε άλλη περίπτωση.

Έπειτα στη `main()` συγκρίνετε τα σημεία `p1` και `p2` και τυπώστε το συμπέρασμα της σύγκρισης.

Αν τα σημεία ενός πολυγώνου φυλάσσονται σε ένα πίνακα από Σημεία, γράψτε μια (εξωτερική) συνάρτηση `perimeter` που να επιστρέφει τη περίμετρο του πολυγώνου (θεωρείστε πως τα διαδοχικά σημεία είναι και σε διαδοχικές θέσεις στο πίνακα).

Έπειτα στη `main()` ορίστε ένα μονοδιάστατο πίνακα από σημεία με μέγεθος 3 (`triangle[3]`) και αναθέστε σε αυτόν τα `p1`, `p2` και `p3`. Στη συνέχεια τυπώστε τη περίμετρο του `triangle`.

3.9 Σε ένα παιχνίδι αντιστοίχησης ο μαθητής θα πρέπει να αντιστοιχεί τα ανακατεμένα στοιχεία της αριστερής στήλης με στοιχεία της δεξιάς στήλης.

Σε C++ δημιουργήστε μια κλάση η οποία να αναπαριστά τέτοια ζεύγη αντιστοίχησης

Στη `main()` ορίστε ένα πίνακα από τέτοια ζεύγη. Τυπώστε τον σωστό πίνακα αντίστοιχής.

Ανακατέψτε τυχαία τα στοιχεία της αριστερής στήλης, καθώς και τα στοιχεία της δεξιάς στήλης.

Ξανατυπώστε το πίνακα.

Παράδειγμα εξόδου:

```
Ordered matching table
              Greece          Athens
            Barbatos          Bridgetown
             Bahrain          Manama
                Fiji          Suva
             Ecuador          Quito

Shuffled matching table
             Bahrain          Suva
            Barbatos          Bridgetown
             Ecuador          Quito
                Fiji          Manama
              Greece          Athens
```

Σημείωση: Κάντε χρήση των συναρτήσεων `rand` και `srand` εισάγοντας τις βιβλιοθήκες `cstdlib` και `ctime`.

Κεφάλαιο 4

Υπερφόρτωση τελεστών και φίλες συναρτήσεις

υπερφόρτωση τελεστών - μονομελείς τελεστές - δυαδικοί τελεστές - φίλες συναρτήσεις - αντικείμενα χωρίς όνομα

Υπερφόρτωση τελεστών

Η C++ υποστηρίζει ένα σύνολο από τελεστές για τους ενσωματωμένους τύπους της, όπως οι +, *, <=, +=, [], <<, …Η χρήση τους όμως σε ορισμένους από εμάς τύπους έχει απροσδιόριστα αποτελέσματα ή/και δεν υποστηρίζεται.

Η υπερφόρτωση τελεστών (operators overloading) επιτρέπει να ορίζουμε τη σημασία των τελεστών όταν τους χρησιμοποιούμε σε δικούς μας τύπους δεδομένων.

Η υπερφόρτωση ενός τελεστή γίνεται με τον ορισμό μιας συνάρτησης τελεστή (operator function). Το όνομα μιας συνάρτησης τελεστή αποτελείται από το προσδιοριστικό `operator` ακολουθούμενο από τον ίδιο το τελεστή. Προηγείται ο τύπος της επιστρεφόμενης τιμής και τέλος τα ορίσματα μέσα σε παρένθεση.

```
1  ... operator@(...);  // @ is the operator
```

Στη C++ μπορούμε να υπερφορτώσουμε τους εξής τελεστές:

```
+       -       *       /       %       ^       &
|       ~       !       =       <       >       +=
=       *=      /=      %=      ^=      &=      |=
<<      >>      >>=     <<=     ==      !=      <=
>=      &&      ||      ++      --      ->*      ,
->      []      ()      new     new[]   delete  delete[]
```

Δεν μπορούμε να υπερφορτώσουμε τους τελεστές:

```
::      scope resolution
.       τελεστής τελεία
.*      επιλογή με τη χρήση δείκτη σε συνάρτηση
```

Επίσης, δεν μπορούμε να ορίσουμε και να υπερφορτώσουμε δικούς μας τελεστές, ούτε να αλλάξουμε την προτεραιότητα των πράξεων.

Υπερφόρτωση Μονομελών Τελεστών

Οι μονομελείς τελεστές (unary operators) είναι οι τελεστές που εφαρμόζονται σε (επηρεάζουν) ένα μόνο αντικείμενο, π.χ. οι ++, --, !, ~ ...

Ένας μονομελείς τελεστής (επιθεματικός ή προθεματικός) μπορεί να οριστεί είτε με μια μη στατική συνάρτηση μέλος που δε δέχεται κανένα όρισμα,

```
1  class FooBar {
2
3     public:
4        ... operator@();
5  };
6
7  ... FooBar::operator@() {
8     // some code here
9  }
```

είτε με μία συνάρτηση μή μέλος που δέχεται ένα όρισμα

```
1  class FooBar {
2        ...
3  };
4
5  ... operator@(FooBar f) {
6     // some code here
7  }
```

Θέτουμε το τύπο που επιστρέφει μια συνάρτηση υπερφόρτωσης ανάλογα με τη χρήση της. Έτσι, αν για παράδειγμα θέλουμε να υπερφορτώσουμε τον προθεματικό τελεστή ++ ώστε να μπορούμε να γράφουμε για την `FooBar` μόνο την έκφραση

```
1  ++foo;   // operator++ doesnt need to return something
```

τότε το πιο πιθανό είναι να ορίσουμε τον επιστρεφόμενο τύπο ως `void`: `void operator++();`

Αν όμως θέλουμε να υποστηρίζει και εκφράσεις ανάθεσης (=)

```
1  FooBar foo2;
2  foo2 = ++foo; // operator++ must return a FooBar object
```

τότε θα πρέπει να επιστρέφει αντικείμενο τύπου FooBar: `FooBar operator++();`

Το πρόγραμμα 4.1 ορίζει μια κλάση FooBar για την οποία υπερφορτώνει το μοναδιαίο τελεστή αύξησης ++ ως πρόθεμα (prefix) και ως επίθεμα (postfix), ορίζοντας αντίστοιχες συναρτήσεις υπερφόρτωσης.

Πρόγραμμα 4.1: Υπερφόρτωση Μονομελών Τελεστών

```
1  // overloadfoobar.cpp
2  // demonstrates binary operators overloading
3  #include <iostream>
4
5  using namespace std;
6
7  class FooBar {
8     private:
9        int data;
10    public:
11       FooBar();
12       FooBar(int d);
13       void print() const;
14
15       // unary operators overloading
16       FooBar operator++();    // increment prefix, ++foo
17       FooBar operator++(int i);  // increment postfix, foo++
18 };
19
20 FooBar::FooBar() {
21    data = 0;
22 }
23
24 FooBar::FooBar(int d) {
25    data = d;
26 }
27
28 void FooBar::print() const {
29    cout << data << endl;
30 }
31
32 // the increment operator (prefix)
33 FooBar FooBar::operator++() {
34    // increments data and then constructs an
35    // unnamed temporary FooBar object and returns it
36    return FooBar(++data);
37 }
38
39 // the increment operator (postfix)
40 FooBar FooBar::operator++(int i){
41    // constructs an unnamed temporary FooBar object,
42    // returns it, and then increments data
43    return FooBar(data++);
44 }
```

```
45
46 int main() {
47    FooBar foo1, foo2;
48    cout<< "foo1: "; foo1.print();   // 0
49    cout<< "foo2: "; foo2.print();   // 0
50
51    foo1 = ++foo2; // operator++() call for foo2
52
53    cout<< "foo1: "; foo1.print();   // 1
54    cout<< "foo2: "; foo2.print();   // 1
55
56    foo1 = foo2++; // operator++(int) call for foo2
57
58    cout<< "foo1: "; foo1.print();   // 1
59    cout<< "foo2: "; foo2.print();   // 2
60
61    return 0;
62 }
```

Υπάρχουν δύο δηλώσεις για την υπερφόρτωση του ++: μία για τη χρήση του ως πρόθεμα (π.χ. ++foo),

```
1 FooBar FooBar::operator ++ ();
```

και μία για τη χρήση του ως επίθεμα (π.χ. foo++)

```
1 FooBar FooBar::operator ++ (int);
```

Η διαφορά τους είναι το ακέραιο όρισμα (int), το οποίο στη συγκεκριμένη περίπτωση δεν είναι πραγματικό. Η C++ το χρησιμοποιεί ως σήμα στον μεταγλωττιστή ώστε να υπερφορτώσει τον τελεστή ως επίθεμα.

Το ίδιο θα ίσχυε και για τον -- τελεστή.

Επισημαίνεται ότι οι δηλώσεις στις γραμμές 36, και 43 (πρόγραμμα 4.1) δημιουργούν ένα νέο αντικείμενο FooBar, που δεν έχει όνομα (nameless object), χρησιμοποιώντας τον δεύτερο κατασκευαστή, και το επιστρέφουν. Αφού επιστραφεί το αντικείμενο αυτό διαγράφεται.

Πιο περιφραστικά θα γράφαμε το εξής:

```
1 FooBar FooBar::operator ++ () {
2    int tmp;
3    tmp = ++data;
4    FooBar result(tmp);
5    return result;
6 }
```

Επίσης, σε αυτή τη περίπτωση το ίδιο αποτέλεσμα θα είχε η χρήση της αυτοαναφοράς, this.

```
1 FooBar FooBar::operator ++ () {
2    ++data;
```

```
3     return *this;
4 }
```

Το πρόγραμμα 4.1 έχει σαν έξοδο:

```
foo1: 0
foo2: 0
foo1: 1
foo2: 1
foo1: 1
foo2: 2
```

Δυαδικοί Τελεστές

Οι δυαδικοί τελεστές (binary operators) είναι οι τελεστές που εφαρμόζονται σε δύο αντικείμενα, π.χ. οι $+$, $-$, $+=$, $>$, $<$, $==$, ...

Ένας δυαδικός τελεστής @ μπορείς να οριστεί είτε με μια μη στατική συνάρτηση μέλος που δέχεται ένα όρισμα,

```
1 class FooBar {
2
3     public:
4         ... operator@(FooBar);
5 };
6
7 ... FooBar::operator@(FooBar f) {
8     // some code here
9 }
```

είτε με μια συνάρτηση μή μέλος που δέχεται δύο ορίσματα.

```
1 class FooBar {
2         ...
3 };
4
5 ... operator@(FooBar a, FooBar b) {
6     // some code here
7 }
```

Όμοια, θέτουμε το τύπο που επιστρέφει μια συνάρτηση υπερφόρτωσης ανάλογα με τη χρήση της.

Το Πρόγραμμα 4.2 ορίζει τους τελεστές $+$ και $+=$, καθώς και τους τελεστές σύγκρισης $<$ και $==$.

Πρόγραμμα 4.2: Υπερφόρτωση Δυμελών Τελεστών

```
1 // overloaddistance.cpp
2 // demonstrates binary operators overloading
```

```
 3
 4  #include <iostream>
 5  using namespace std;
 6
 7  class Distance {
 8     private:
 9        int meters;
10        int centimeters;
11     public:
12        Distance();
13        Distance(int m, int cm);
14        void print() const;
15        int calculateCentimeters() const;
16        Distance operator+=(const Distance& d);
17  };
18
19  Distance::Distance() {
20     meters = 0;
21     centimeters = 0;
22  }
23
24  Distance::Distance(int m, int cm) {
25     meters = m;
26     centimeters = cm;
27  }
28
29  void Distance::print() {
30     cout << meters << "meters, " << centimeters << "centimeters" <<endl;
31  }
32
33  int Distance::calculateCentimeters() const {
34     return meters*100 + centimeters;
35  }
36
37  Distance Distance::operator+=(const Distance& d) {
38     meters += d.meters;
39     centimeters += d.centimeters;
40     return Distance(meters, centimeters);
41  }
42
43  Distance operator+(const Distance& a, const Distance& b) {
44     Distance d = a;
45     return d += b;
46  }
47
48  bool operator<(const Distance& a, const Distance& b) {
49     return (a.calculateCentimeters() < b.calculateCentimeters()) ?
50        true : false;
51  }
52
```

```
53 bool operator==(const Distance& a, const Distance& b) {
54    return (a.calculateCentimeters() == b.calculateCentimeters()) ?
55       true : false;
56 }
57
58 int main() {
59    Distance dist1(10, 5);
60    Distance dist2(2, 4);
61    Distance dist3;
62
63    cout<<"dist1: "; dist1.print();
64    cout<<"dist2: "; dist2.print();
65    cout<<"dist3: "; dist3.print();
66
67    cout<<"Perform dist3 = dist1 + dist2 " << endl;
68    dist3 = dist2 + dist1; // operator+(dist1, dist2) call
69    cout<<"dist3: "; dist3.print();
70
71    if (dist1 < dist2) // operator<(dist1, dist2) call
72       cout << "dist1 is less than dist2" << endl;
73    else
74       cout << "dist1 is greater than dist2" << endl;
75
76    if (dist1 == dist2) // operator==(dist1, dist2) call
77       cout << "dist1 equals dist2" << endl;
78    else
79       cout << "dist1 differs from dist2" << endl;
80 }
```

Παρατηρείστε πως τελεστές που δεν αλλάζουν το αντικείμενο που τους καλεί έχουν οριστεί ως συναρτήσεις μη μέλη. Αυτό δεν επιβάλλεται από τη γλώσσα, αλλά συνίσταται ως καλή τακτική.

Επίσης, ο ορισμός νέων τελεστών μπορεί να γίνει με τη χρήση ήδη ορισμένων τελεστών (όπως στο Πρόγραμμα 4.2 ο $+$ ορίζεται με βάση τον $+=$, γραμμή 45). Προσοχή όμως σε κυκλικές αναφορές.

Επιπλέον θυμηθείτε τη κλήση κατά αναφορά, η οποία στη C++ υλοποιείται είτε με τη χρήση δεικτών είτε με τη χρήση αναφορών. Για να αποφύγουμε τις αντιγραφές μεγάλων αντικειμένων μπορούμε να δηλώσουμε τις συναρτήσεις ώστε να δεχονται ορίσματα αναφορές (Πρόγραμμα 4.2, γραμμή 37, 40). Στη περίπτωση των συναρτήσεων υπερφόρτωσης δεν μπορούμε να χρησιμοποιήσουμε δείκτες, επειδή δεν είναι δυνατόν να αλλάξει ο ορισμός της σημασίας ενό τελεστή που εφαρμόζεται σε δείκτη.

Επίσης η επιστροφή μια αναφοράς

Τέλος, τα ορίσματα της υπερφορτωμένης συνάρτησης, όπως και σε κάθε συνάρτηση, δηλώνουν τις εκφράσεις που επιτρέπονται. Αν για παράδειγμα θέλαμε να μπορούμε να προσθέτουμε έναν ακέραιο σε αντικείμενα τύπου Distance, θα έπρεπε επιπλέον να ορίζαμε και τις:

```
1 Distance operator+(const Distance& d, int i); // d + i
2 Distance operator+(int i, const Distance& d); // i + d
```

Το πρόγραμμα 4.2 έχει σαν έξοδο:

```
dist1: 10meters, 5centimeters
dist2: 2meters, 4centimeters
dist3: 0meters, 0centimeters

Perform dist3 = dist1 + dist2
dist3: 12meters, 9centimeters

dist1 is greater than dist2
dist1 differs from dist2
```

Συναρτήσεις μετατροπής

Η μετατροπή τύπων (casting) είναι ένας άλλος τύπος υπερφόρτωσης.

Οι συναρτήσεις μετατροπής (conversion operator) είναι συναρτήσεις μέλη που ορίζουν τη μετατροπή από το τρέχων τύπο (X) σε ένα άλλο (T):

```
1  X
2  ::operator T();
```

Το όνομα μιας συνάρτησης μετατροπής αποτελείται από το προσδιοριστικό `operator` ακολουθούμενο από το όνομα του τύπου.

Η συνάρτηση μετατροπής μπορεί να καλεστεί ρητά

```
1  Distance dist;
2  double dl;
3  dl = static_cast<double> (dist);
```

ή μέσω του τελεστή ανάθεσης (αυτόματη μετατροπή τύπου).

```
1  dl = dist;
```

Το Πρόγραμμα 4.3 ορίζει τη συνάρτηση μετατροπής αντικειμένων τύπου `Distance` σε πραγματικούς αριθμούς (`double`).

Πρόγραμμα 4.3: Συναρτήσεις μετατροπής

```
1  // convertionoperator.cpp
2  // demonstrates convertion operator functions
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  class Distance {
8     private:
9        int meters;
10       int centimeters;
```

```
11     public:
12        Distance();
13        Distance(int m, int cm);
14        operator double() const; // conversion operator
15  };
16
17  Distance::Distance() {
18     meters = 0;
19     centimeters = 0;
20  }
21
22  Distance::Distance(int m, int cm) {
23     meters = m;
24     centimeters = cm;
25  }
26
27  Distance::operator double() const {
28     float decadal_part = static_cast<double>(centimeters)/100;
29     float int_part = static_cast<double>(meters);
30     return int_part + decadal_part;
31  }
32
33  int main() {
34     double dl;
35     Distance dist(10, 3);
36
37     dl = static_cast<double>(dist); // uses operator double(),
38                  // same as dl = dist;
39     cout << dl << endl; // prints 10.03
40
41     return 0;
42  }
```

Ευχή και κατάρα

Η υπερφόρτωση των τελεστών είναι ένα χαρακτηριστικό της C++ που μας επιτρέπει να χρησιμοποιούμε βολικούς και συμβατικούς συμβολισμούς για το χειρισμό αντικειμένων. Από την άλλη, η κακή χρήση τους μπορεί να οδηγήσει στη παρερμηνεία του κώδικα.

Γι' αυτό καλό είναι

- να ορίζουμε τους τελεστές με τρόπο που να μιμούνται το συμβατικό τρόπο χρήσης τους από τους ενσωματωμένους τύπους
- παρόμοιοι τελεστές να έχουν παρόμοια συμπεριφορά (πχ. οι + και +=)
- να ορίζουμε νέες συναρτήσεις αντί για να υπερφορτώνουμε τελεστές όπου υπάρχουν αμφιβολίες για τη συσχέτιση των πράξεων που γίνονται με τον τελεστή.

Φίλες συναρτήσεις

Οι φίλες συναρτήσεις (friend functions) είναι συναρτήσεις που έχουν πρόσβαση στα ιδιωτικά μέλη κλάσεων.

Δηλώνουμε ρητά μια συνάρτηση ως φίλη χρησιμοποιώντας το προσδιοριστικό `friend` μπροστά από τον τύπο της συνάρτησης είτε στο ιδιωτικό είτε στο δημόσιο μέρος μιας δήλωσης τάξης.

Ορίζουμε τη φίλη συνάρτηση εκτός από τη δήλωση της κλάσης, χωρίς να επαναλαμβάνουμε το προσδιοριστικό `friend`.

Μια περίπτωση χρήσης φίλων συναρτήσεων είναι οι συναρτήσεις που θέλουμε να μπορούν να κάνουν πράξεις με ιδιωτικά δεδομένα μέλη δύο ή περισσότερων μη συσχετιζόμενων Κλάσεων.

Πρόγραμμα 4.4: Φίλες Συναρτήσεις ως γέφυρα δύο μη συσχετιζόμενων Κλάσεων

```
1  // friend_function.cpp
2  // demonstrates friend functions
3
4  #include <iostream>
5  #include <cstring>
6
7  using namespace std;
8
9  class Beta;
10
11 class Alpha {
12    private:
13       int data;
14    public:
15       Alpha(int d) { //1-arg constructor
16          data = d;
17       };
18
19    // friend function
20    friend int friendFunction(Alpha a, Beta b);
21 };
22
23
24 class Beta {
25    private:
26       int data;
27    public:
28       Beta(int d) { //1-arg constructor
29          data = d;
30    };
31
32    // friend function
33    friend int friendFunction(Alpha a, Beta b);
34 };
35
36 // friend function definition
37 int friendFunction(Alpha a, Beta b) {
```

```
38    // friendFunction has access to private members a.data and b.data
39    return a.data + b.data;
40 }
41
42 int main() {
43
44    Alpha a = 1;
45    Beta b = 2;
46
47    // friend function call
48    cout << friendFunction(a, b) << endl; // prints 3
49
50    return 0;
51 }
```

Επιπλέον, οι δυαδικοί τελεστές είναι μια συνηθισμένη περίπτωση χρήσης φίλων συναρτήσεων.

Πρόγραμμα 4.5: Φίλες Συναρτήσεις για πιο ευέλικτες συναρτήσεις υπερφόρτωσης

```
1  // friend_functions2.cpp
2  // using friend functions to increase the
3  // flexibility of the overloaded operators
4  //
5  // NOTE: to avoid excessive copying (although Foo has a small
6  // representation), functions are  declared to take reference args
7
8  #include <iostream>
9  using namespace std;
10
11 class Foo {
12    private:
13       int data;
14    public:
15       Foo(int d); // 1-arg constructor
16       void print();
17    // operator overload as friend
18    friend Foo operator+(const Foo& foo1, const Foo& foo2);
19 };
20
21 Foo::Foo(int d) {
22    data = d;
23 }
24
25 void Foo::print() {
26    cout << data << endl;
27 }
28
29 Foo operator+(const Foo& foo1, const Foo& foo2) {
30    return foo1.data + foo2.data;
```

```
31 }
32
33 int main() {
34    Foo f1 = 1 + 2; // Foo(1) + Foo(2)
35    f1.print(); // 3
36
37    Foo f2 = 5; // Foo(5)
38    Foo f3 = 5 + f2; // Foo(5) + f2
39    f3.print(); // 10
40
41    Foo f4 = f1 + 2; // f1 + Foo(2)
42    f4.print(); // 5
43
44    Foo f5 = f1 + f2;
45
46    return 0;
47 }
```

Ακόμη, οι φίλες συναρτήσεις επιτρέπουν ένα πιο κατανοητό τρόπο κλήσης συναρτήσεων. Για παράδειγμα, μερικοί προγραμματιστές προτιμούν το συμβολισμό `invert(m)` για την αντιστροφή ενός αντικειμένου τύπου `matrix`, αντί για τον `m.invert()`.

```
1  class Matrix {
2     ...
3     friend void invert(Matrix& m);
4  };
5
6  int main() {
7     Matrix m;
8     ...
9     invert(m);
10    ...
11 }
```

Ασκήσεις

Για της δηλώσεις των συναρτήσεων υπερφόρτωσης συμβουλευτείτε τον πίνακα 4.1[1]:

4.1 Μια ημερομηνία αποτελείται από ημέρες, μήνες, έτη. Σε C++ ορίστε μια κλάση `Date` που να αναπαριστά ημερομηνίες. Υπερφορτώστε τους μονομελείς τελεστές $++$ και $--$ ώστε να αυξάνουν και να μειώνουν την ημερομηνία κατά μία μέρα αντίστοιχα.

4.2 Στο πρότυπο χρώματος `RGB` το κόκκινο, το πράσινο και το μπλε φως συνδυάζονται παράγοντας μια ποικιλία χρωμάτων.

[1] `http://www.cplusplus.com/doc/tutorial/classes2/`

Ένα χρώμα στο πρότυπο χρώματος RGB μπορεί να περιγραφεί με το προσδιορισμό του πόσο κάθε ένα από το κόκκινο, πράσινο και μπλε χρώματα περιλαμβάνεται. Οι τιμές χρώματος μπορούν να γραφτούν ως ακέραιοι αριθμοί στην κλίμακα 0 έως 255.

Για παράδειγμα, αν όλα τα χρώματα έχουν τη μέγιστη τιμή, τότε έχουμε το άσπρο $(255, 255, 255)$, το αντίθετο συμβαίνει στο μαύρο $(0, 0, 0)$. Ομοίως το μπλε $(0, 0, 255)$, το κόκκινο $(255, 0, 0)$, το πράσινο $(0, 255, 0)$, το κίτρινο $(255, 255, 0)$...

Το αρνητικό (negative) ενός χρώματος ορίζεται ως η διαφορά του από το άσπρο: $\sim (\mathrm{r}, \mathrm{g}, \mathrm{b}) = (255, 255, 255) - (\mathrm{r}, \mathrm{g}, \mathrm{b})$

Η ανάμιξη δύο χρωμάτων ορίζει ένα νέο κάθε χρώμα του οποίου είναι ο μέσος όρος των χρωμάτων που αναμίχτηκαν: $(\mathrm{r}_1, \mathrm{g}_1, \mathrm{b}_1) + (\mathrm{r}_2, \mathrm{g}_2, \mathrm{b}_2) = ((\mathrm{r}_1 + \mathrm{r}_2)/2, (\mathrm{g}_1 + \mathrm{g}_2)/2, (\mathrm{b}_1 + \mathrm{b}_2)/2)$

Προσαρμόζουμε τη φωτεινότητα ενός χρώματος πολλαπλασιάζοντας κάθε χρώμα με ένα δεκαδικό αριθμό (brightness) που ανήκει στο διάστημα 0 έως 1. $(\mathrm{r}, \mathrm{g}, \mathrm{b}) * \mathrm{brightness} = ((\mathrm{r} * \mathrm{brightness}), (\mathrm{g} * \mathrm{brightness}), (\mathrm{b} * \mathrm{brightness}))$

Σε C++, δημιουργείστε μία κλάση που να αναπαριστά χρώματα στο RGB. Υπερφορτώστε τους τελεστές $\sim$, $+$, και $*$ με το τρόπο που ορίζονται παραπάνω. Φτιάξτε `main()` που να ελέγχει τη λειτουργία των τελεστών αυτών.

4.3 Κάθε κλάσμα (fraction) έχει έναν ακέραιο αριθμητή και έναν ακέραιο παρονομαστή (που δε μπορεί να είναι μηδέν).

Σε C++ δημιουργείστε μία κλάση που να αναπαριστά κλάσματα.

Υπερφορτώστε το μονομελή τελεστή (unary operator) ὥστε να αντιστρέφει το κλάσμα.

Υπερφορτώστε τους διμελής τελεστές (binary operators) $+$, $-$, $/$, $*$ ώστε να υλοποιούν τις βασικές πράξεις μεταξύ των κλασμάτων.

Φτιάξτε `main()` που να ελέγχει τη λειτουργία των τελεστών αυτών.

Παράδειγμα εξόδου:

```
~(1/3) = (3/1)
(5/6) + (7/9) = (87/54)
(5/6) - (7/9) = (3/54)
(5/6) * (7/9) = (35/54)
(5/6) / (7/9) = (45/42)
```

4.4 Στα μαθηματικά, ένας μιγαδικός αριθμός είναι ένας αριθμός που αποτελείται από το πραγματικό μέρος και το φανταστικό μέρος: $\mathrm{a} + \mathrm{bi}$ όπου a και b είναι πραγματικοί αριθμοί.

Σε C++ δημιουργείστε μια κλάση που να αναπαριστά μιγαδικούς αριθμούς. Υπερφορτώστε τη πράξη της πρόσθεσης $+$ με τρόπο ώστε να υποστηρίζονται οι πράξεις:

```
complex3 = complex1 + complex2;
complex3 = complex1 + 2; // add 2 to the real part of complex1
complex3 = 4 + complex1; // add 4 to the real part of complex1
```

Φτιάξτε `main()` που να ελέγχει τη λειτουργία των τελεστών αυτών.

4.5 Στη `Distance` κλάση (Πρόγραμμα 4.2) υπερφορτώστε τον δυαδικό τελεστή $*$ ώστε δύο αποσταστάσεις να μπορούν να πολλαπλασιαστούν. Ορίστε την ως φίλη συνάρτηση και με τρόπο που να επιτρέπονται οι εκφράσεις:

```
dist1 = 3 * dist2;
dist1 = dist2 * 39;
dist1 = dist2 * dist3;
```

Expression	Operator	Member function
@a	+ - * & ! ~ ++ --	A::operator@()
a@	++ --	A::operator@(int)
a@b	+ - * / % ^ & | < > == != <= >= << >> && || ,	A::operator@(B)
a@b	= += -= *= /= %= ^= &= |= <<= >>= []	A::operator@(B)
a(b, c...)	()	A::operator()(B, C...)
a->x	->	A::operator->()

όπου @ ο προς υπερφόρτωση τελεστής, και a, b, c αντικείμενα κλάσεων A, B, C.

Πίνακας 4.1: Δηλώσεις συναρτήσεων τελεστή

Κεφάλαιο 5

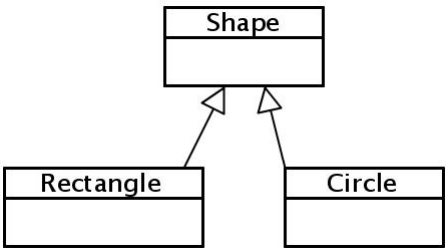
Παράγωγες Κλάσεις

κληρονομικότητα - ιεραρχία κλάσεων - προστατευμένα μέλη - βασική κλάση - παράγωγη κλάση - ιδεατές συναρτήσεις - πολλαπλή κληρονομικότητα - αφηρημένες κλάσεις

Κληρονομικότητα

Συχνά έννοιες που αναπαριστούμε με κλάσεις στον κώδικά μας σχετίζονται μεταξύ τους ιεραρχικά.

Για παράδειγμα, αν θέλουμε να μοντελοποιήσουμε έννοιες όπως ο Κύκλος και το Ορθογώνιο, ανακαλύπτουμε πως αυτές σχετίζονται μεταξύ τους, είναι και τα δύο επίπεδα γεωμετρικά Σχήματα. Έχοντας κοινή την έννοια του Σχήματος, μοιράζονται κοινές ιδιότητες και χαρακτηριστικά, αλλά συγχρόνως η κάθε μία έχει επιπλέον στοιχεία που τη διακρίνουν από την άλλη. Π.χ., όλα τα Σχήματα έχουν ένα χρώμα, ένα κέντρο, κ.α., αλλά ο Κύκλος έχει ακτίνα, το Ορθογώνιο έχει ύψος και πλάτος, κ.ο.κ.



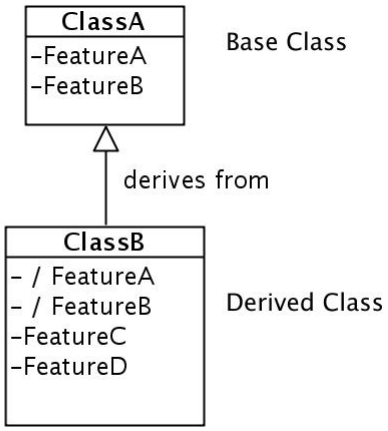
Σχήμα 5.1: Κληρονομικότητα

Ακόμα περισσότερο, αν τα επίπεδα γεωμετρικά Σχήματα είναι μια κατηγορία σχημάτων, ο Κύκλος είναι μια υπό-κατηγορία αυτής. Το ίδιο και το Ορθογώνιο.

Στη C++ μπορούμε να εκφράσουμε τέτοιες ιεραρχικές σχέσεις μεταξύ δύο ή περισσότερων κλάσεων.

Μια κλάση (subclass - υπο-κλάση) μπορεί να παραχθεί από μια άλλη κλάση (superclass - υπερ-κλάση), υιοθετώντας από αυτή τις κοινές ιδιότητες και χαρακτηριστικά, και έπειτα προσθέτοντας τα στοιχεία που τη διακρίνουν.

Στη C++, η πρώτη κλάση καλείται παράγωγη (derived class), η δεύτερη καλείται βασική κλάση (base class), και η συσχέτισή τους καλείται κληρονομικότητα (inheritance)(Σχήμα 5.2).



Σχήμα 5.2: Κληρονομικότητα

Μετά τις κλάσεις, η κληρονομικότητα είναι το βασικότερο χαρακτηριστικό του αντικειμενοστραφή προγραμματισμού.

Συμβάλει στη σταδιακή οικοδόμηση των κλάσεων, και κατά συνέπεια και του συστήματός μας.

Επιπλέον, οι κλάσεις σε μια ιεραρχία κλάσεων λειτουργούν ως δομικές μονάδες για την υλοποίηση πιο εξειδικευμένων κλάσεων. Η κληρονομικότητα δηλαδή υποστηρίζει το προγραμματισμό με βηματική βελτίωση του κώδικα (incremental refinement).

Θέλουμε να επεκτείνουμε κλάσεις μας δημιουργώντας παράγωγες κλάσεις γιατί

- έτσι δεν φορτώνουμε τις κλάσεις μας με πολλά στοιχεία που θα τα ``κουβαλάμε'' σε κάθε περίπτωση είτε χρειάζονται είτε όχι.
- έτσι επαναχρησιμοποιούμε κώδικα που έχουμε ήδη δημιουργήσει για άλλες περιπτώσεις (reusability), κάτι που μας σώζει χρόνο και χρήματα, και αυξάνει την αξιοπιστία του κώδικά μας.
- δεν θέλουμε να αλλάξουμε μια τρέχουσα κλάση μιας και την χρησιμοποιούμε ήδη σε άλλες περιπτώσεις και είμαστε ευτυχισμένοι με αυτή, την έχουμε τεστάρει (testing) και απο-σφαλματώσει (debugging) καλά.
- μπορεί να μην έχουμε πρόσβαση στην βασική κλάση, π.χ. όταν θέλουμε να επεκτείνουμε κλάσεις που ανήκουν σε μια βιβλιοθήκη.

Δήλωση Παράγωγης Κλάσης

Για να ορίσουμε μια κλάση ως παράγωγη, τότε μετά το όνομά της χρησιμοποιούμε το τελεστή `:` ακολουθούμενο από το προσδιοριστή πρόσβασης και το όνομα της βασικής κλάσης:

```
1 class derived_class : access_specifier base_class
```

Όταν ο προσδιοριστής πρόσβασης είναι ο `public`, η παράγωγη κλάση κληρονομεί όλα τα μέλη της παράγωγης διατηρώντας τους ίδιους προσδιοριστές πρόσβασης.

Στο παρακάτω παράδειγμα, ορίζεται η κλάση `Foo` και έπειτα η παράγωγη `Bar`:

```
1  class Foo {
2     ...
3  };
4
5  class Bar : puplic Foo {
6     ...
7  };
```

Όταν ο προσδιοριστής πρόσβασης είναι ο `protected`, η παράγωγη κλάση κληρονομεί τα δημόσια `public` και προστατευμένα (`protected`) μέλη της βασικής σαν να είχαν δηλωθεί στην ίδια ως `protected`, και ιδιωτικά (`private`) ως (`private`)

Όταν ο προσδιοριστής πρόσβασης είναι ο `private`, η παράγωγη κλάση κληρονομεί τα μέλη της βασικής σαν να είχαν δηλωθεί στην ίδια ως `private`.

Προσβασιμότητα και δημόσια Κληρονομικότητα

Ο παρακάτω πίνακα (Πίνακας 5.1) διευκρινίζει τη προσβασιμότητα των δεδομένων και των συναρτήσεων της Βασικής Κλάσης από την ίδια, την Παράγωγη Κλάση (δημόσια κληρονομικότητα) και τις εξωτερικές συναρτήσεις, όπως π.χ. είναι η `main()`.

Προσδιοριστής Πρόσβασης των μελών της Βασικής Κλάσης	Προσβάσιμα από την Βασική Κλάση	Προσβάσιμα από την Παράγωγη Κλάση	Προσβάσιμα από εξωτερικές συναρτήσεις
`public`	ναι	ναι	ναι
`protected`	ναι	ναι	όχι
`private`	ναι	**όχι**	όχι

Πίνακας 5.1: Προσβασιμότητα και δημόσια κληρονομικότητα

Παρατηρείστε πως η παράγωγη κλάση δεν εχει πρόσβαση στα ιδιωτικά μέλη της βασικής. Μπορεί να ακούγεται περίεργο, αλλά το αντίθετο θα επέτρεπε σ' έναν προγραμματιστή να αποκτήσει πρόσβαση στα ιδιωτικά μέλη μιας κλάσης απλά δημιουργώντας μια παράγωγη αυτής.

Συσχετίσεις is-a και has-a

Η **δημόσια** κληρονομικότητα έχει τη σημασία της is-a (είναι) συσχέτισης. Αν μια παράγωγη κλάση `D` κληρονομεί δημόσια την κλάση `B`, υπονοείται τόσο στον μεταγλωττιστή όσο και στους αναγνώστες του κώδικα ότι κάθε αντικείμενο τύπου `D` είναι επίσης αντικείμενο τύπου `B`, αλλά όχι το αντίθετο. Ακόμη υπονοείται πως η `B` αναπαράσταση είναι πιο γενική από τη `D`, και πως η `D` αναπαριστά μια πιο ειδική έννοια από την `B`. Επιπλέον, συμπεραίνεται πως όπου μπορούμε να χρησιμοποιήσουμε αντικείμενο τύπου `B`, μπορούμε να χρησιμοποιήσουμε αντικείμενο τύπου `D`, γιατί το `D` is-a (είναι) αντικείμενο τύπου `B`. Το αντίθετο όμως δεν ισχύει.

Για παράδειγμα, γνωρίζουμε πως κάθε Μαθητής (`Student`) είναι Άνθρωπος (`Person`), ενώ κάθε Άνθρωπος δεν είναι Μαθητής.

```
class Person { ... };

class Student : puplic Person {
   ...
};
```

Από την άλλη, η σύνθεση (composition, συνώνυμοι όροι layering, containment, aggregation, embedding) είναι μια συσχέτιση ανάμεσα σε κλάσεις και υπάρχει όταν αντικείμενα ενός τύπου περιέχουν αντικείμενα του άλλου.

Για παράδειγμα, ένας Άνθρωπος (`Person`) αποτελείται, εκτός των άλλων, από μια Διεύθυνση (`Address`), και αντικείμενα τύπου ΑριθμόςΤηλεφώνου (`PhoneNumber`). Η σύνθεση έχει τη σημασία της has-a (έχει) συσχέτισης.

Έτσι, ένας Άνθρωπος λέμε πως έχει Διεύθυνση, Αριθμό Τηλεφώνου σπιτιού και Αριθμό Τηλεφώνου γραφείου.

```
class Address { ... };

class PhoneNumber { ... };

class Person {
   ...
   Address address;
   PhoneNumber homePhoneNumber;
   PhoneNumber officePhoneNumber;
};
```

Κατασκευαστές Παράγωγης Κλάσης

Οι κατασκευαστές της παράγωγης κλάσης μπορούν να λαμβάνουν υπόψη τους τα κληρονομημένα δεδομένα μέλη και κατασκευαστές της βασικής.

Στο Πρόγραμμα 5.1 δείτε πως δηλώνονται και ορίζονται οι κατασκευαστές των παράγωγων κλάσεων.

Πρόγραμμα 5.1: Κατασκευαστές Παράγωγης Κλάσης

```
// inheritance.cpp
// demonstrates public inheritance and constructors for the derived class
#include <iostream>

using namespace std;

// base class
class Foo {
   protected:
      int foodata;
   public:
      Foo();      // no-args constructor
```

```
13        Foo(int fd);   // 1-arg constructor
14  };
15
16  Foo::Foo() {
17     cout << "Foo no-args constructor call" << endl;
18     foodata = 0;
19  }
20
21  Foo::Foo(int fd) {
22     cout << "Foo 1-arg constructor call" << endl;
23     foodata = fd;
24  }
25
26  // derived class
27  class Bar : public Foo {
28     private:
29        int bardata;
30     public:
31        Bar();         // no-args constructor
32        Bar(int fd, int bd); // 2-args constructor
33  };
34
35  Bar::Bar() : Foo() {
36     cout << "Bar no-args constructor call" << endl;
37     bardata = 0;
38  }
39
40  Bar::Bar(int fd, int bd) : Foo(fd) {
41     cout << "Bar 2-args constructor call" << endl;
42     bardata = bd;
43  }
44
45  int main() {
46     cout << "Creating foo1" << endl;
47     Foo foo1; cout << endl;
48
49     cout << "Creating bar1" << endl;
50     Bar bar1; cout << endl;
51
52     cout << "Creating foo2" << endl;
53     Foo foo2(10); cout << endl;
54
55     cout << "Creating bar2" << endl;
56     Bar bar2(20, 10); cout << endl;
57
58     return 0;
59  }
```

Ο μόνος τρόπος για να περάσουμε τιμές στους κατασκευαστές της βασικής κλάσης είναι μέσω της λίστας

μελών αρχικοποίησης (member initialization list): μετά τον προσδιορισμό των παραμέτρων του κατασκευαστή, εισάγουμε το τελεστή : και στη συνέχεια, χωρισμένα με κόμματα, τα ονόματα των κλάσεων με τα τυχών ορίσματά τους (Πρόγραμμα 5.1, γραμμές 35, 40).

Έτσι, όταν λέμε `Bar bar1;` ο μεταγλωττιστής δημιουργεί ένα αντικείμενο τύπου `Bar`. Στη συνέχεια καλεί τον `Bar()` κατασκευαστή για να το αρχικοποιήσει. Αυτός με τη σειρά του καλεί τον `Foo()` κατασκευαστή, πριν από οποιαδήποτε άλλη δήλωση μπορεί ο ίδιος να περιέχει. Τέλος η `Bar()` αρχικοποιεί τη `bardata` με το `0`.

Γι αυτό το Πρόγραμμα 5.1 έχει έξοδο:

```
Creating foo1
Foo no-args constructor call

Creating bar1
Foo no-args constructor call
Bar no-args constructor call

Creating foo2
Foo 1-arg constructor call

Creating bar2
Foo 1-arg constructor call
Bar 2-args constructor call
```

Επιπλέον, μπορούμε να χρησιμοποιήσουμε τη λίστα μελών αρχικοποίησης για να περάσουμε τιμές και στα δεδομένα μέλη της κλάσης (π.χ. `bardata(0)`).

```
1  Bar::Bar() : Foo(), bardata(0) {
2  }
3
4  Bar::Bar(int fd, int bd) : Foo(fd), bardata(bd) {
5  }
```

Κάτι τέτοιο μερικές φορές επιβάλλεται. Στο παρακάτω παράδειγμα, δεν υπάρχει άλλος τρόπος για να αρχικοποιήσουμε το στατικό μέλος της κλάσης `somedata`, και το αντικείμενο `otherdata` (δεδομένου ότι το τελευταίο ορίζει τον default κατασκευαστή του `AnotherClass()`):

```
1  class SomeClass {ι
2     prvate:
3        const int somedata;
4        AnotherClass otherdata;
5        ...
6     public:
7        SomeClass();
8        ...
9  };
10
11 SomeClass::SomeClass() : somedata(0), otherdata() {
12
```

```
13 }
```

Υπερκάλυψη συναρτήσεων μελών της βασικής κλάσης

Στη κληρονομικότητα, κάθε παράγωγη κλάση έχει πρόσβαση στα προστατευμένα και δημόσια μέλη της βασικής κλάσης.

Κανόνας της C++ είναι πως όταν μια συνάρτηση ορίζεται και στην βασική και στην παράγωγη, τότε όταν κληθεί από αντικείμενο της παράγωγης κλάσης εκτελείται αυτή της παράγωγης. Λέμε πως η συνάρτηση της παράγωγης κλάσης υπερταιρεί, υπερκαλύπτει (overrides) τη συνάρτηση της βασικής κλάσης.

Στο Πρόγραμμα 5.2 η συνάρτηση `print()` της `DerivedClass` υπερκαλύπτει την `print()` της `BaseClass`.

Πρόγραμμα 5.2: Υπερκάλυψη συναρτήσεων μελών

```
1  // override.cpp
2  // demonstrates that derived class functions override base class functions
3  // that have the same name, violating the is-a rule of public inheritance
4  #include <iostream>
5
6  using namespace std;
7
8  // base class
9  class Foo {
10    public:
11       void print() const {
12          cout << "Base class printing" << endl;
13       }
14 };
15
16 // derived class
17 class Bar : public Foo {
18    public:
19       // overrides Foo::print() const
20       void print() const {
21          cout << "Derived class printing" << endl;
22       }
23 };
24
25 int main() {
26    Foo foo;
27    Bar bar;
28
29    foo.print();   // Foo::print() call
30    bar.print();   // Bar::print() call
31    bar.Foo::print(); // Foo::print() call
32
33    return 0;
34 }
```

Αν θέλουμε να εκτελεστεί αυτή της βασικής θα πρέπει να το δηλώσουμε ρητά:

```
1  bar.Foo::print();
```

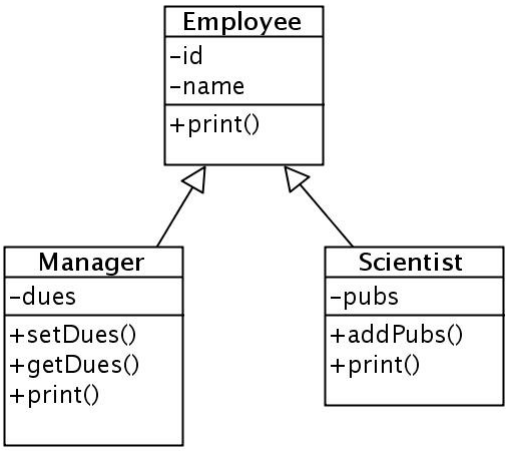
Γι αυτό το Πρόγραμμα 5.2 έχει σαν έξοδο:

```
Base class printing
Derived class printing
Base class printing
```

Προσέξτε πως η υπερκάλυψη των συναρτήσεων μ' αυτό το τρόπο σπάει τον κανόνα της είναι (is-a) συσχέτισης της δημόσιας κληρονομικότητας.

Ένα ολοκληρωμένο παράδειγμα δημόσιας κληρονομικότητας

Σκεφτείτε πως θέλουμε να μοντελοποιήσουμε τους υπαλλήλους μιας εταιρίας ερευνών. Όλοι οι υπάλληλοι έχουν αριθμό υπαλλήλου και όνομα. Οι διευθυντές είναι υπάλληλοι, και επιπλέον είναι μέλη σε γκολφ club στο οποίο πληρώνουν μια συνδρομή. Οι ερευνητές είναι υπάλληλοι που διαθέτουν έναν αριθμό από δημοσιεύσεις.



Με τη χρήση της κληρονομικότητας ορίζουμε τις κατηγορίες υπαλλήλων Μάνατζερ (`Manager`) και Ερευνητές (`Scientist`) ως παράγωγες κλάσης της βασικής κλάσης των Υπαλλήλων (`Employee`).

Πρόγραμμα 5.3: Ένα ολοκληρωμένο παράδειγμα δημόσιας κληρονομικότητας

```
1  // employee.cpp
2  // demonstrates public inheritance, base and derived class
3  #include <iostream>
4  #include <string>
5
6  using namespace std;
7
8  const int MAX_NAME_SIZE = 20;
```

```
 9
10 // Base class
11 class Employee {
12    protected:
13       int id;  // employee number
14       string name;   // employee full name
15    public:
16       Employee(); // default constructor
17       Employee(int eid, const string& ename);   // 2-arg constructor
18       void print() const;
19 };
20
21 Employee::Employee() {
22    id = 0;
23    name = "";
24 }
25
26 Employee::Employee(int eid, const string& ename) {
27    id = eid;
28    name = ename;
29 }
30
31 void Employee::print() const {
32    cout << "Employee id " << id << ", Name: " << name << endl;
33 }
34
35
36 // Derived class
37 class Manager : public Employee {
38    private:
39       float dues; // golf dues
40    public:
41       Manager(); // default constructor
42       Manager(int eid, const string& ename, float d); // 3-arg constructor
43       float getDues() const;
44       void setDues(float mdues);
45       void print() const;  // overrides the Employee::print()
46 };
47
48 Manager::Manager() : Employee() {
49    dues = 0;
50 }
51
52 Manager::Manager(int eid, const string& ename, float d)
53    : Employee(eid, ename) {
54    dues = d;
55 }
56
57 float Manager::getDues() const {
58    return dues;
```

```
59 }
60
61 void Manager::setDues(float mdues) {
62    dues = mdues;
63 }
64
65 void Manager::print() const {
66    Employee::print();
67    cout << "Golf dues: " << dues << endl;
68 }
69
70
71 // Derived class
72 class Scientist : public Employee {
73    private:
74       int pubs; // num of publications
75    public:
76       Scientist();   // default constructor
77       Scientist(int eid, const string& ename, int p); // 3-arg constructor
78       int addPubs(int p); // adds p number of publications to pubs
79       void print() const; // overrides the Employee::print()
80 };
81
82 Scientist::Scientist() : Employee() {
83    pubs = 0;
84 }
85
86 Scientist::Scientist(int eid, const string& ename, int p)
87    : Employee(eid, ename) {
88    pubs = p;
89 }
90
91 /* adds p to the current number of publications,
92  * and returns the updated pubs */
93 int Scientist::addPubs(int p) {
94    return pubs += p;
95 }
96
97
98 void Scientist::print() const {
99    Employee::print();
100    cout << "Number of publications: " << pubs << endl;
101 }
102
103
104 int main() {
105    Manager m(102, "King", 2000);
106    m.print(); // Manager::print();
107
108    Scientist s(103, "Alby", 300);
```

```
109     s.print();// Scientist::print();
110     cout << "2 more Alby's papers have been approved. Now he has "
111        << s.addPubs(2) << " publications." << endl;
112
113     return 0;
114  }
```

Το πρόγραμμα 5.3 έχει σαν έξοδο:

```
Employee id 102, Name: King
Golf dues: 2000

Employee id 103, Name: Alby
Number of publications: 300

2 more Alby's papers have been approved. Now he has 302 publications.
```

Δείκτης σε υποκλάση

Επειδή μια παράγωγη κλάση (π.χ. η `Manager`, Πρόγραμμα 5.3) είναι και βασική (π.χ. `Employee`, δλδ. ένας Μάνατζερ είναι Υπάλληλος), σε ένα δείκτη τύπου βασικής κλάσης μπορεί να ανατεθεί η διεύθυνση ενός αντικειμένου τύπου παράγωγης κλάσης. Αντίθετα, ένας δείκτης τύπου παράγωγης κλάσης δε μπορεί να δείχνει σε αντικείμενο βασικής.

```
1  void f() {
2     Manager m;
3     Employee *eptr;   // pointer to an employee
4
5     eptr = &m;  // eptr points to manager m
6           // correct: every manager is also an employee
7
8     eprt->setDues(1000); // sets golf dues to 1000
9     ...
10 }
```

Ιδεατές συναρτήσεις

Οι ιδεατές συναρτήσεις (virtual functions) μας επιτρέπουν να ορίζουμε συναρτήσεις στην βασική κλάση που θα μπορούν να ορίζονται ξανά σε κάθε παράγωγη κλάση. Σε κάθε περίπτωση, ο μεταγλωττιστής θα καλεί για κάθε αντικείμενο την κατάλληλη παραλλαγή της (πολυμορφισμός - polymorphism).

Δηλώνουμε μια συνάρτηση μέλος ως ιδεατή χρησιμοποιώντας το προσδιοριστικό `virtual`.

```
1  class BaseClass {
2          ..
3          public:
```

```
4       ...
5       virtual void print() const; // a virtual function
6   }
```

Μια συνάρτηση της παράγωγης με το ίδιο όνομα και ορίσματα με την ιδεατή υπερκαλύπτει (overrides) την εικονική.

```
1   class DerivedClass : public BaseClass {
2           ..
3           public:
4       ...
5       virtual void print() const; // overrides BaseClass::print()
6   }
```

Στο παρακάτω παράδειγμα δείτε τη διαφορά ιδεατών και μη ιδεατών συναρτήσεων που κληρονομεί και αντικαθιστά μια παράγωγη κλάση.

Παρατηρείστε πως ο δείκτης σε αντικείμενο της παράγωγης καλεί την επαναπροσδιορισμένη ιδεατή συνάρτηση της παράγωγης.

Πρόγραμμα 5.4: Ιδεατές Συναρτήσεις

```
1   // nonvirtual_and_virtual_functions.cpp
2   // demonstrates how the derived class overrides a non-virtual function
3   // violating the is-a rule of public inheritance
4
5   #include <iostream>
6
7   using namespace std;
8
9   class Base {
10     public:
11        void nvf() {
12           cout << "Hello from Base::nvf()" << endl;
13        }
14        virtual void vf() {
15           cout << "Hello from Base::vf()" << endl;
16        }
17  };
18
19  class Derived : public Base {
20     public:
21        void nvf() {
22           cout << "Hello from Derived::nvf()" << endl;
23        }
24        virtual void vf() {
25           cout << "Hello from Derived:vf()" << endl;
26        }
27  };
```

```
28
29 int main() {
30    Base base;
31    Derived derived;
32
33    base.nvf();
34    base.vf();
35
36    derived.nvf();
37    derived.vf();
38    // so far everything is expected.
39
40    Base *bptr = &base;
41    Derived *dptr = &derived;
42
43    bptr->nvf();   // Base::nvf()
44    bptr->vf(); // Base::vf()
45
46    dptr->nvf();   // Derived::nvf()
47    dptr->vf(); // Derived::vf()
48    // again, pretty much expected
49
50    Base *bptr2 = &derived; // a base ptr can point to objects of the derived
51
52    bptr2->nvf();  // nvf() is not virtual, a call to nvf() will always invoke
53          // the implementation associated with the pointer type
54          // -- bptr2' type is Base. Thus it's a Base::nvf() call.
55
56    bptr2->vf();   // vf() is virtual, when refering to an instance of the
57          // Derived by a pointer (or reference) to the Base class,
58          // the correct implementation will be resolved, i.e. the
59          // implementation of the type of the instance. Thus it's
60          // a Derived::vf() call.
61
62    return 0;
63 }
```

Ο σκοπός των ιδεατών συναρτήσεων είναι να δηλώσουμε πως οι παράγωγες κλάσεις κληρονομούν τη διασύνδεση (interface) της συνάρτησης αλλά και μια εξ ορισμού υλοποίησή της, προδιαθέτοντας πως ο προγραμματιστής μπορεί να την επαναπροσδιορίσει.

Πολλαπλή Κληρονομικότητα

Μια κλάση μπορεί να παράγεται από περισσότερες από μία κλάσεις. Τότε λέμε πως έχουμε πολλαπλή κληρονομικότητα (multiple inheritance):

```
1 class derived_class : access_specifier base_class1, public base_class2, ... {
```

```
2     ...
3  }
```

Στην περίπτωση αυτή η κλάση κληρονομεί τα μέλη και τις μεθόδους όλων των βασικών κλάσεων που δηλώθηκαν.

Αφηρημένες κλάσεις

Η έννοια της κληρονομικότητας αποτελείται από δύο διαφορετικά μέρη: την κληρονομικότητα διασύνδεσης (interface) μιας συνάρτησης και την κληρονομικότητα υλοποίησης (implementation) μιας συνάρτησης.

Πολλές φορές θέλουμε η παράγωγη κλάση να κληρονομεί μόνο την διασύνδεση μιας συνάρτησης μέλος. Αυτό το πετυχαίνουμε με τις αφηρημένες κλάσεις (abstract classes). Αφηρημένες κλάσεις είναι αυτές που περιέχουν μία ή περισσότερες γνήσιες εικονικές συναρτήσεις (pure virtual function) και μια εικονική συνάρτηση είναι γνήσια όταν η αρχική της τιμή είναι `= 0`.

Παρακάτω, η κλάση `Shape` είναι μια αφηρημένη κλάση. Στο πρόγραμμά μας δεν μπορούμε να δημιουργήσουμε αντικείμενα τύπου `Shape` (δεν θα είχε νόημα), ενώ οι παράγωγες κλάσεις της κληρονομούν μόνο την διασύνδεση την `draw()` συνάρτησης και μπορούν να δώσουν τη δική τους υλοποίηση.

```
1  class Shape {
2     ...
3     public:
4        // a pure virtual function: all shapes are drawable but Shape
5        // class cannot provide any reasonable default implementation
6        virtual void draw() const = 0;
7  };
8
9  class Rectangle : public Shape {
10    ...
11    public:
12       // define here how Rectangles are drawned
13       virtual void draw() const { ... };
14 }
```

Μια γνήσια εικονική συνάρτηση που δεν ορίζεται στην παράγωγη τάξη παραμένει γνήσια εικονική συνάρτηση, και η παράγωγη κλάση μετατρέπεται και αυτή σε αφηρημένη.

Είναι πολύ σημαντική η χρήση των αφηρημένων τάξεων μιας και αυτές παρέχουν μια διασύνδεση με την οποία πρέπει να δομηθούν συμβατές με αυτή κλάσεις, προστατεύοντας (κρύβοντας) παράλληλα λεπτομέρειες του κώδικα.

Ασκήσεις

5.1 Σε C++ να υλοποιήσετε κλάσεις που να αναπαριστούν φοιτητές του τμήματος Πληροφορικής και Λογιστικής.

Οι φοιτητές έχουν αριθμό εγγραφής, όνομα, έτος εγγραφής και αριθμό περασμένων μαθημάτων.

Ορίστε συναρτήσεις μέλη που να επιστρέφουν αληθές αν ο φοιτητή μπορεί να πάρει πτυχίο, δεδομένου πως οι φοιτητές του τμήματος πληροφορικής παίρνουν πτυχίο στα 40 περασμένα μαθήματα και του Λογιστικής στα 38.

Υπερφορτώστε τον τελεστή σύγκρισης > για τους φοιτητές, θεωρώντας πως μεγαλύτερος είναι ο πιο παλιός φοιτητής.

Να χρησιμοποιήσετε κληρονομικότητα για μια πιο συνοπτική υλοποίηση.

Φτιάξτε `main()` και φοιτητές από τα δύο τμήματα. Εκλέξτε τη δυνατότητα να πάρουν πτυχίο. Συγκρίνετέ τους και τυπώστε αντίστοιχο μήνυμα.

Παράδειγμα εξόδου:

```
12345, Alice, 38 passed courses, 1998 year of registration
12346, Wally, 38 passed courses, 1999 year of registration

Alice can graduate
Wally cannot graduate

Alice is older student than Wally
```

5.2 Ένα παιχνίδι (game) περιλαμβάνει ένα σύνολο από Χαρακτήρες (`GameCharacter`). Κάθε Χαρακτήρας έχει ένα όνομα και έναν ακέραιο που αναπαριστά την υγεία του χαρακτήρα, δηλαδή το πόση ζημία (συνήθως από την άποψη του φυσικού τραυματισμού) ένας χαρακτήρας μπορεί να αντισταθεί.

Σε C++, υλοποιείστε κλάση που να αναπαριστά τέτοιους Χαρακτήρες, ορίζοντας παράλληλα έναν κατασκευαστή που να αρχικοποιεί το όνομα του Χαρακτήρα και να θέτει την υγεία ίση με το 100, και έναν δεύτερο που να αρχικοποιεί όλα τα χαρακτηριστικά.

Οι Πολεμιστές (`Warrior`) είναι Χαρακτήρες παιχνιδιών. Πέρα από το όνομα και την υγεία έχουν έναν ακέραιο που αναπαριστά την φυσική τους δύναμη.

Υλοποιείστε κλάσης που να αναπαριστά Πολεμιστές, χρησιμοποιώντας κληρονομικότητα. Ορίστε έναν κατασκευαστή που να αρχικοποιεί το όνομά του Πολεμιστή και να θέτει την υγεία ίση με 100 και τη δύναμη με 50, και έναν κατασκευαστή που να αρχικοποιεί όλα τα χαρακτηριστικά. Δημιουργείστε συνάρτηση μέλος που να τυπώνει τα χαρακτηριστικά του Πολεμιστή.

Στη `main()` δημιουργείστε 2 Πολεμιστές, `warrior1` και `warrior2` με τον πρώτο κατασκευαστή, ενσωματώνοντας δικά σας ονόματα. Τυπώστε τα στοιχεία των Πολεμιστών.

Ένας Χαρακτήρας έχει την ιδιότητα να πίνει φίλτρα `drinkHealthPotion(int p)` τα οποία ανανεώνουν την υγεία τους κατά `p`. Στη `main()` δώστε στον `warrior2` να πιει ένα μαγικό φίλτρο.

Υπερφορτώστε τον μοναδιαίο τελεστή ++ (ως επίθεμα) ώστε να αυξάνει κατά ένα τη δύναμη του Πολεμιστή. Στη `main()` αυξήστε τη δύναμη του `warrior1`.

Ένας Πολεμιστής μπορεί να μονομαχίσει με έναν άλλο Πολεμιστή. Κάτι τέτοιο όμως δε θα είναι φρόνιμο αν ο δεύτερος Πολεμιστής είναι πιο δυνατός. Ορίστε συνάρτηση μέλος `shouldFight` που να επιστρέφει αληθές όταν η δύναμη του Πολεμιστή είναι μεγαλύτερη σε σχέση με έναν άλλο και ψευδές για το αντίθετο. Στη `main()` ελέγξτε το αν ο `warrior1` θα πρέπει να πολεμήσει τον `warrior2`.

Παράδειγμα εξόδου:

```
Warrior1: 100 health, 50 fight ability
Warrior2: 100 health, 50 fight ability
```

```
warrior 2 drink a health potion
Warrior2: 120 health, 50 fight ability

Increasing fight ability for warrion 1
Warrior1: 100 health, 51 fight ability

warrior 1: start the fight
```

5.3 Μια Ηλεκτρονική Συσκευή (`ElectronicGadget`) έχει ένα κωδικό μοντέλου (αλφαριθμητικό), βάρος και τιμή.

Σε C++ υλοποιείστε μια κλάση που να αναπαριστά Ηλεκτρονικές Συσκευές. Ορίστε έναν κατασκευαστή που να αρχικοποιεί όλα τα παραπάνω χαρακτηριστικά.

Ένα Mp3 Player (`Mp3Player`) είναι μια Ηλεκτρονική Συσκευή. Επιπλέον χαρακτηρίζεται από τον συνολικό χώρο αποθήκευσης που διαθέτει, και τον ελεύθερο χώρο αποθήκευσης.

Υλοποιείστε κλάση που αναπαριστά Mp3 Players χρησιμοποιώντας κληρονομικότητα. Ορίστε κατασκευαστή που να αρχικοποιεί όλα τα χαρακτηριστικά του. Ορίστε συνάρτηση μέλος που να τυπώνει τα στοιχεία του Mp3 Player.

Στη `main()` δημιουργείστε δύο Mp3 Players , `mp3Player1` και `mp3Player2`, ενσωματώνοντας δικά σας στοιχεία. Τυπώστε τα στοιχεία των δύο Player.

Μπορούμε να αντιγράψουμε τραγούδια από ένα Mp3 Player σε ένα άλλο, όταν υπάρχει αντίστοιχος ελεύθερος χώρος. Ορίστε συνάρτηση μέλος `canCopySongsFrom` η οποία να επιστρέφει αληθές όταν το Mp3 Player έχει ελεύθερο χώρο αποθήκευσης μεγαλύτερο του χώρου των τραγουδιών του δεύτερου Mp3 Player, και ψευδές για το αντίθετο.

Στη `main()` ελέγξτε το αν το `mp3Player2` μπορεί να αντιγράψει όλα τα τραγούδια του `mp3Player1`.

Παράδειγμα εξόδου:

```
Mp3 Player 1: 12.3 weight, 50 price, 120 disk space, 100 free disk space
Mp3 player 2: 15.3 weight, 80 price, 150 disk space, 50 free disk space

Can Mp3 player 2 copy songs from Mp3 player 1?
Yes, Mp3 player 2 has enough disk space
```

5.4 Ορίστε τη βασική κλάση `BaseClass`

```
class BaseClass {
        public:
                virtual void printme() const {
                        cout << "Printing from the Base Class" << endl;
                }
}
```

Έπειτα ορίστε δύο παράγωγες και σε κάθε μία ορίστε την αντίστοιχη `printme()`. Στην `main()` δημιουργείστε ένα αντικείμενο για κάθε παράγωγη και καλέστε μέσα από αυτά την `printme()`. Ορίστε δείκτες αντικειμένων των παράγωγων κλάσεων και καλέστε μέσα από αυτά την `printme()`.

5.5 Με δεδομένες τις τάξεις `Circle`, `Square` και `Triangle` που παράγονται από την τάξη `Shape` (Σχήμα), ορίστε μια συνάρτη `totalArea()`, που να δέχεται ως όρισμα ένα πίνακα από Σχήματα και να επιστρέφει το άθροισμα των εμβαδόν όλων των Σχημάτων του πίνακα. Θα χρειαστεί να ορίσετε κατάλληλες εικονικές συναρτήσεις.

5.6 Με δεδομένες τις τάξεις `Circle`, `Square` και `Triangle` που παράγονται από την τάξη `Shape`, ορίστε μια συνάρτη `intersect()`, που να δέχεται δύο ορίσματα τύπου `Shape*` και να προσδιορίζει αν τα δύο τμήματα επικαλύπτονται. Θα χρειαστεί να ορίσετε κατάλληλες εικονικές συναρτήσεις.

Κεφάλαιο 6

Συνδυαστικές ασκήσεις

Ασκήσεις

6.1 Μια Τάξη αποτελείται από το Δάσκαλο και τους Μαθητές. Επιπλέον, κατά τη διάρκεια του χρόνου διδάσκονται στη τάξη Μαθήματα. Οι Μαθητές βαθμολογούνται για τα Μαθήματα. Σε C++ φτιάξτε πρόγραμμα που να μπορεί να τυπώνει αναλυτικές καταστάσεις για κάθε τάξη.

Παράδειγμα εξόδου:

```
Class: 12th Grade - L1
Teacher: John Keating

            SocialStudies  Mathematics  Science  LanguageArts   Avg
Todd             7.5           10         10         8          8.9
Neil            10              7.5        8         9          8.6
Charlie          5              5          6         7          5.6
...


Class Average: 7.7
```

6.2 Σύμφωνα με το σύστημα κρυπτογράφησης του Καίσαρα, κάθε γράμμα του Κειμένου αντικαθίσταται με το γράμμα που βρίσκονται 3 θέσεις μετά στο αλφάβητο (π. χ. το μήνυμα ``a secret message'' κρυπτογραφείται στο ``d vhfuhw phvvdjh'').

Αν τα μηνύματα που θέλουμε να στέλνουμε από έναν Κόμβο σε έναν άλλο θέλουμε να είναι κρυπτογραφημένα με αυτό το τρόπο, σε C++ ορίζοντας κατάλληλες κλάσεις και συναρτήσεις, φτιάξτε πρόγραμμα που να επιτρέπει τέτοιου είδος ανταλλαγές.

6.3

Γλωσσάρι

argument	όρισμα
assignment	ανάθεση
base class	βασική κλάση
binary operator	διμελής τελεστής
binary operators	δυαδικοί τελεστές
build-in	ενσωματομένος
call by reference	κλήση κατά αναφορά
call by value	κλήση κατά τιμή
constructor	κατασκευαστής
container	αποδέκτες, περιέχοντες
declaration	δήλωση
definition	ορισμός
denstructor	καταστροφέας
derived class	παράγωγη κλάση
encapsulation	ενθυλάκωση, data hinding
expression	παράσταση, έκφραση, κάθε διάταξη από μεταβλητές, σταθερές και τελεστές που ορίζει έναν υπολογισμό. Μια πρόταση μπορεί να περιέχει πολλές εκφράσεις
friend function	φίλες συναρτήσεις
header file	αρχείο επικεφαλίδας
identifier	αναγνωριστικό, όνομα
inheritance	κληρονομικότητα
iterator	επαναλήπτης
loop	βρόχος
manipulator	χειριστής
operator overloading	υπερφόρτωση τελεστών

parameter	παράμετρος
prefix	πρόθεμα, π.χ. ++a
qualifier	διευκρινιστικό
relational operators	συσχετιστικοί τελεστές
standart output stream	καθιερωμένο ρεύμα εξόδου
statement	πρόταση, εντολή. Οι προτάσεις είναι εντολές προς τον υπολογιστή προκειμένου να κάνει κάτι. Οι προτάσεις τελειώνουν πάντα με ελληνικό ερωτηματικό
unary operator	μονομελής τελεστής
virtual function	ιδεατή συνάρτηση

Βιβλιογραφία

[1] cplusplus. The c++ resources network. URL http://www.cplusplus.com.

[2] R. Lafore. The Waite Group's Object-oriented programming in C++. Sams Publishing, third edition, 1999.

[3] S. Meyers. Effective C++ : 55 Specific Ways to Improve Your Programs and Designs. Addison-Wesley Professional, third edition, May 2005. URL http://www.aristeia.com.

[4] sgi. Standard template library programmer's guide. URL http://www.sgi.com/tech/stl/.

[5] B. Stroustrup. The C++ Programming Language. Addison-Wesley Professional, third edition, February 2000. URL http://www.research.att.com/~bs/C++.html.

Παράρτημα Α

Καθιερωμένοι Χειριστές

Οι καθιερωμένοι χειριστές (standard manipulators) είναι εντολές για το ρεύμα εξόδου που μορφοποιούν την έξοδο με διάφορους τρόπους (βλέπε επίσης [5] Ενότητα 21.4).

iomanip

smanip setw(int n) θέτει το πλάτος της εκτύπωσης ισο με n

smanip setfill(int c) θέτει τον χαρακτήρα συμπλήρωσης ίσο με c

smanip setprecision(int n) θέτει το πλήθος των δεκαδικών ψηφίων που θα εμφανίζονται ίσο με n

smanip setbase(int b) θέτει την αριθμητική βάση της εξόδου των ακεράιων ίση με b

iostream

ios_base& scientific(ios_base&) εμφανίζει τους αριθμούς κινητής υποδιαστολής με τη μορφή d.ddddddEdd

ios_base& fixed(ios_base&) εμφανίζει τους αριθμούς κινητής υποδιαστολής με τη μορφή ddd.dd

ios_base& dec(ios_base&) εμφανίζει τους αριθμούς στο δεκαδικό σύστημα

ios_base& oct(ios_base&) εμφανίζει τους αριθμούς στο οκταδικό σύστημα

ios_base& hex(ios_base&) εμφανίζει τους αριθμούς στο δεκαεξαδικό σύστημα

ios_base& left(ios_base&) θέτει αριστερή στοίχιση

ios_base& right(ios_base&) θέτει δεξιά στοίχιση

ios_base& uppercase(ios_base&) τυπώνει κεφαλαία

ios_base& lowercase(ios_base&) τυπώνει μικρά

Πρόγραμμα Α.1: Παραδείγματα με χειριστές

```
 /**
  * Standard Manipulators in formatting situations
  *
  * @see B. Stroustroup, The C++ Programming Language 3rd Edition, Sec 21.4
  */

#include <iostream>
#include <iomanip>

using namespace std;

int main() {
   cout << left << setw(8) << "width: "
      << right << setw(5)<< setprecision(3) << 4.32415 << endl
      << left << setw(8)   << "height: "
      << right << setw(5)<< setprecision(3) << 7.24265 << endl;

   cout << "dec: " << 64 << endl
      << "oct: " << oct << 64 << endl
      << "hex: " << hex << 64 << endl;

   return 0;
}
```