Εισαγωγή στον Προγραμματισμό

Μέρος 1ο: Αντικείμενα και κλάσεις

Εξάμηνο Σπουδών: 3ο

Κωδικός Μαθήματος: 343

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Στόχοι

- Καλή γνώση βασικών αρχών προγραμματισμού
- Καλή γνώση βασικών αρχών αντικειμενοστρεφή προγραμματισμού
- Ικανότητα προγραμματισμού στη γλώσσα C++

Χρόνος ενασχόλησης

- **Διαλέξεις:** Τρίτη 14:00-17:00
- **Εργαστήριο:** Τετάρτη 11:00-13:00 (προαιρετικά)

- **Τουλάχιστον** 4 ώρες μελέτη την εβδομάδα!

- Η παρακολούθηση των εργαστηρίων και η ολοκλήρωση των ασκήσεων είναι η καλύτερη προετοιμασία για το διαγώνισμα.

Εργαστήρια

- Περίπου 8 εβδομαδιαία εργαστήρια/ασκήσεις
- Σκοπός: η εμπέδωση της ύλης, η εξάσκηση και η λύση απορίων

- **Υπεύθυνοι Εργαστηρίου:**

  Σωτήριος Κοντογιάννης, Ε.Ε.ΔΙ.Π., Τμήμα Μαθηματικών
  Κωνσταντίνα Τζουβάρα, Ε.Τ.Ε.Π., Τμήμα Μαθηματικών

Εργαστηριακές ασκήσεις

- **Εκφώνηση:** ανακοινώνεται στις αρχές κάθε εβδομάδας
- **Λύση:** μέχρι την Κυριακή 23:59 στον αντίστοιχο σύνδεσμο του ecourse

- Κάθε άσκηση βαθμολογείται με **παρουσία** ή **απουσία**:
  - **Παρουσία:** στοιχειώδες ενασχόληση με τα αντίστοιχα ερωτήματα
  - **Απουσία:** δεν έχει αποσταλεί ή δεν εμφανίζει επαρκή ενασχόληση

- Απαγορεύονται αυστηρά οι αντιγραφές (όχι όμως οι συνεργασίες)

- Για τη συμμετοχή στις εξετάσεις, χρειάζονται **τουλάχιστον 5 παρουσίες**
  - Αφορά όλους τους φοιτητές (ακόμα και παλαιότερα έτη)

Αξιολόγηση

- **Test εργαστηρίου:** 30% (μετά τις διακοπές των Χριστουγέννων)
- **Διαγώνισμα:** 70%

- **Τελική Βαθμολογία:** (με τη προϋπόθεση των $\geq$ 5 παρουσιών)

70% βαθμός γραπτής εξέτασης + 30% βαθμός τεστ εργαστηρίου

Συγγράμματα

- **Σύγγραμματα που διανέμονται:**
  - R. Lafore, Αντικειμενοστρεφής προγραμματισμός με τη C++, Εκδόσεις Κλειδάριθμος.
  - Ν. Χατζηγιαννάκης, Η γλώσσα C++ σε βάθος, Εκδόσεις Κλειδάριθμος.
  - W. Savitch, Πλήρης C++, Εκδόσεις Τζιόλα.
  - L. Jesse, Πλήρες εγχειρίδιο της C++, Εκδόσεις Α. Γκιούρδα.
  - H. Deitel and P. Deitel, C++ Προγραμματισμός 6η Εκδοση, Εκδόσεις Μ. Γκιούρδας.
  - J. V. Guttag, Υπολογισμοί και Προγραμματισμός με την Python, Εκδόσεις Κλειδάριθμος.
  - D. Schneider, Εισαγωγή στον Προγραμματισμό με την Python, Εκδόσεις Μ. Γκιούρδας.
  - Σ. Καλαφατούδης, Γ. Σταμούλης, Προγραμματισμός με την Python, Εκδόσεις Νέων Τεχνολογιών.

Θέματα που θα καλύψουμε στα πλαίσια του μαθήματος

- **C++** (11 διαλέξεις)
  - Αντικείμενα και κλάσεις
  - Συναρτήσεις
  - Πίνακες - Ταξινόμηση - Αναζήτηση
  - Συμβολοσειρές
  - Είσοδος - Έξοδος
  - Δείκτες
  - Πρότυπα
  - Η καθιερωμένη βιβλιοθήκη προτύπων STL
- **Python** (2 διαλέξεις)
  - Ιστορικό
  - Βασικό συντακτικό
  - Τύποι - Τελεστές - Ροή ελέγχου
  - Συναρτήσεις
  - Κλάσεις
  - Εργαλεία

Μέρος 1°:
Εισαγωγή

Στυλ Προγραμματισμού

- Διαδικασιακός προγραμματισμός
  - Δεδομένα (μεταβλητές, πίνακες, εγγραφές) - βασικά δομικά στοιχεία ενός προγράμματος
  - Κώδικας - εντολές προς τον υπολογιστή σχετικά με το πως να χειριστεί τα δεδομένα.
  - Στηρίζεται σε καλά ορισμένες δομές ελέγχου
    - Ενσωματωμένοι τύποι, τελεστές, εντολές, συναρτήσεις, δομές
    - Δομές ελέγχου (if, switch), δομές επανάληψης (π.χ. while), εκφράσεις και εκχωρήσεις

- Αντικειμενοστραφής προγραμματισμός
  - Εστιάζει στον σχεδιασμό και την υλοποίηση ιεραρχιών κλάσεων
  - Βασίζεται στην έννοια του αντικειμένου
    - Τα αντικείμενα ομαδοποιούν δεδομένα και τις λειτουργίες πάνω σε αυτά τα δεδομένα
    - Επιτρέπουν την απόκρυψη των δεδομένων

Στυλ Προγραμματισμού

- Διαδικασιακός προγραμματισμός — C
  - Δεδομένα (μεταβλητές, πίνακες, εγγραφές) - βασικά δομικά στοιχεία ενός προγράμματος
  - Κώδικας - εντολές προς τον υπολογιστή σχετικά με το πως να χειριστεί τα δεδομένα.
  - Στηρίζεται σε καλά ορισμένες δομές ελέγχου
    - Ενσωματωμένοι τύποι, τελεστές, εντολές, συναρτήσεις, δομές
    - Δομές ελέγχου (if, switch), δομές επανάληψης (π.χ. while), εκφράσεις και εκχωρήσεις

- Αντικειμενοστραφής προγραμματισμός — C++
  - Εστιάζει στον σχεδιασμό και την υλοποίηση ιεραρχιών κλάσεων
  - Βασίζεται στην έννοια του αντικειμένου
    - Τα αντικείμενα ομαδοποιούν δεδομένα και τις λειτουργίες πάνω σε αυτά τα δεδομένα
    - Επιτρέπουν την απόκρυψη των δεδομένων

Επίδειξη

- `Empty.cpp`
- `HelloWorld.cpp`
- `Square.cpp`
- `Fraction.cpp`

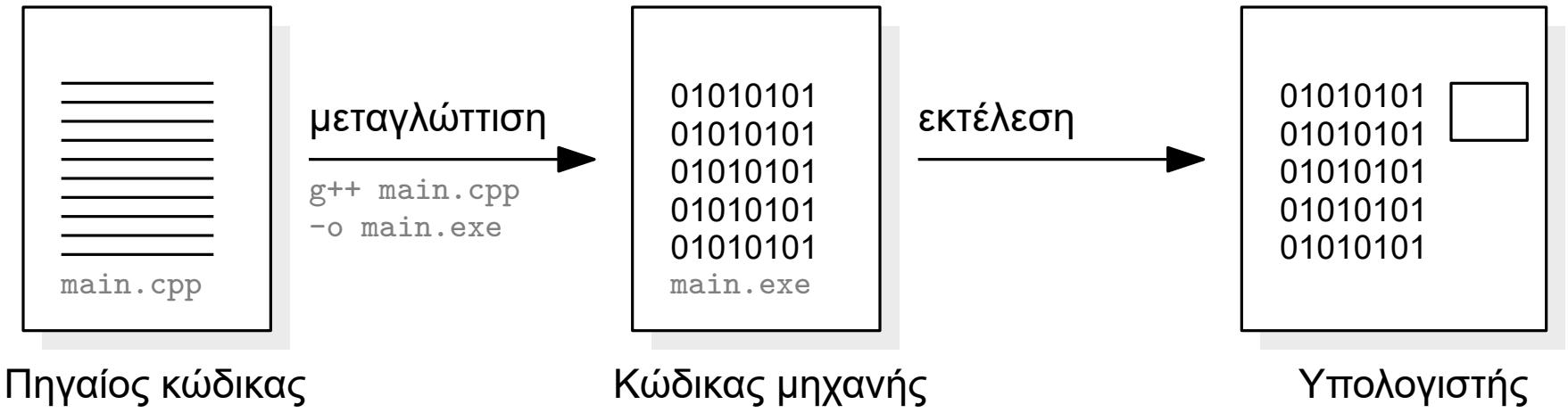
Επίδειξη

- `Empty.cpp` Το μικρότερο C++ πρόγραμμα
- `HelloWorld.cpp` Το δημοφιλέστερο C++ πρόγραμμα
- `Square.cpp` Ένα παράδειγμα διαδικασιακού προγραμματισμού
- `Fraction.cpp` Ένα παράδειγμα αντικειμενοστραφούς προγραμματισμού

- Τι είδαμε;
  - Μια γλώσσα προγραμματισμού: C++
  - Ένα πρόγραμμα: σε C++
  - Ένα περιβάλλον ανάπτυξης: Visual Code Studio

Μεταγλώττιση και εκτέλεση

- **Πηγαίος κώδικας:** Συνήθως γράφεται σε κάποιο περιβάλλον ανάπτυξης (όπως το Visual Studio Code που είδαμε)
- **Μεταγλωττιστής** [compiler]: Ελέγχει τον πηγαίο κώδικα για συντακτικά λάθη
- **Κώδικας μηχανής:** Παράγεται από τον εκάστοτε μεταγλωττιστή σε δυαδική μορφή (υπάρχουν αρκετοί διαθέσιμοι μεταγλωττιστές ανάλογα με το λειτουργικό, π.χ., mingw)

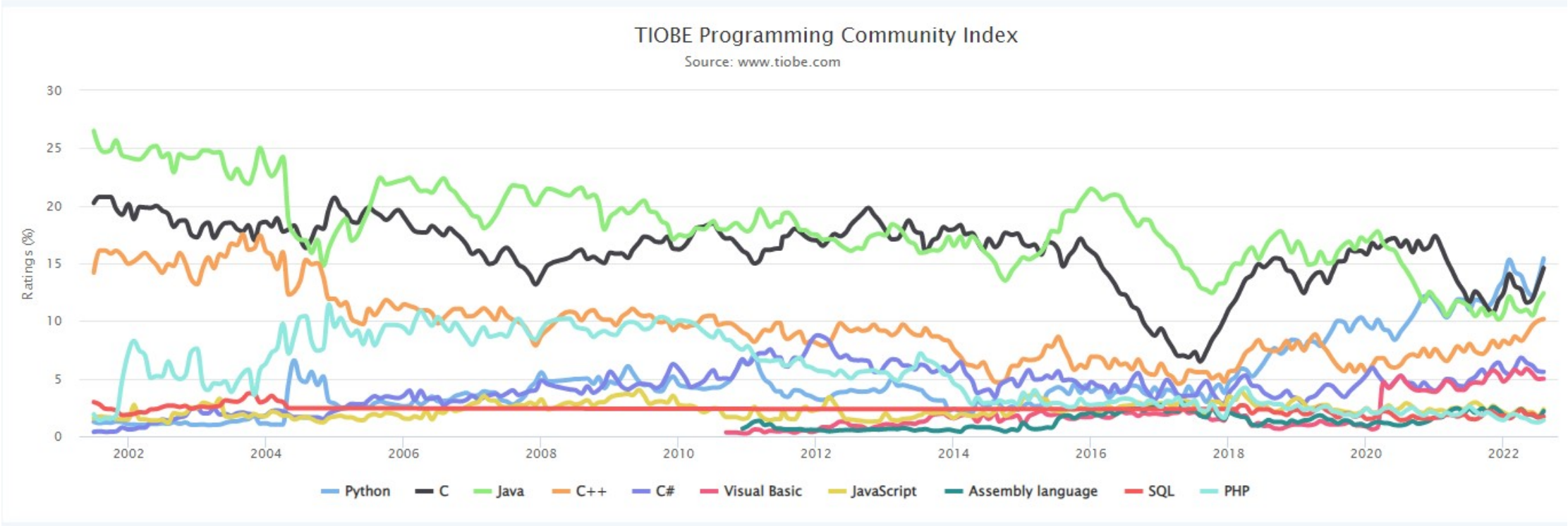


C++

- **B.C.** Γλώσσες όπως FORTRAN, COBOL, ALGOL, PL/I και άλλες.
- **1970:** Οι Brian Kernigham και Dennis Ritchie αναπτύσσουν τη C.
  Η γλώσσα που αποτέλεσε την έμπνευσή τους ονομαζόταν “B”.
- **1980:** Ο Bjarne Stroustrup δημιουργεί τη “C με Κλάσεις”.
- **1995:** Η Επιτροπή ANSI εκδίδει το σχέδιο Προτύπου C++.
- **1998:** Υιοθετείται το επίσημο πρότυπο για τη C++.

- **Γενικά:** Η C++ αποτελεί υπερσύνολο της C παρέχοντας επιπλέον τις κλάσεις

Γιατί C++;



https://www.tiobe.com/tiobe-index/

Visual Studio Code

- Το Visual Studio Code είναι ένα (σχετικά απλό) προγραμματιστικό περιβάλλον
- Παρέχεται δωρεάν από την Microsoft
- Υποστηρίζει πολλές γλώσσες προγραμματισμού (C++, Html, PHP, Java, Python).
- Για την μεταγλώττιση και εκτέλεση προγραμμάτων C++ θα χρειαστείτε:
  - Να εγκαταστήσετε αρχικά ένα μεταγλωττιστή C++
    https://www.mingw-w64.org/downloads/
  - Να εγκαταστήσετε το περιβάλλον Visual Studio Code
    https://code.visualstudio.com/
  - Να εγκαταστήσετε τα πρόσθετα “C/C++ extension for VS Code” και “Code Runner”
    https://code.visualstudio.com/docs/cpp/config-mingw
- Στη σελίδα του μαθήματος στο `ecourse.uoi.gr` θα βρείτε αναλυτικό οδηγό εγκατάστασης τους.

Πως ορίζεται και χρησιμοποιείται μια κλάση σε ένα πρόγραμμα;

- **Δήλωση της κλάσης:** επιλογή του τι θα αποθηκεύουν (πεδία ή μέλη δεδομένων) και πως θα συμπεριφέρονται (μέθοδοι ή μέλη συναρτήσεων) τα αντικείμενα της κλάσης.
- **Ορισμός των μεθόδων ή μελών συναρτήσεων:** παρέχεται υλοποίηση για κάθε μέθοδο της κλάσης.
- **Χρήση της κλάσης για τη δημιουργία αντικειμένων:** δήλωση νέων στιγμιοτύπων αντικειμένων της κλάσης όπως δηλώνονται και οι απλές μεταβλητές.
  - Μια κλάση είναι ένας οριζόμενος από τον χρήστη τύπος
  - Ο προγραμματιστής μπορεί να δημιουργεί ένα ή περισσότερα αντικείμενα μιας κλάσης

```
//Class Fraction
#include<iostream>
using namespace std;

class Fraction {
private:
  int numerator;
  int denominator;

public:
  Fraction() {
    numerator = 0;
    denominator = 1;
  }
  double getValue() {
   return numerator/denominator;
  }
  ...
};
```

```
int main() {
  Fraction f;
  cout << f.getValue() << endl;
}
```

Part01/Fraction.cpp

Η ανατομία μίας κλάσης

- Το σώμα της κλάσης χωρίζεται σε δύο μέρη:
  - ιδιωτικό (private) Παρέχει τη διεπαφή της κλάσης
  - δημόσιο (public) Λεπτομέρειες της υλοποίησης
- Τα δημόσια μέλη αποτελούν τη διασύνδεση για τα αντικείμενα της κλάσης
- Τα ιδιωτικά μέλη είναι προσβάσιμα μόνο από συναστήσεις-μέλη της κλάσης

```
class ClassName {

private:
    Ιδιωτικά πεδία
    Ιδιωτικοί κατασκευαστές
    Ιδιωτικές μέθοδοι

public:
    Δημόσια πεδία
    Δημόσιοι κατασκευαστές
    Δημόσιες μέθοδοι

};
```

- Σημειώσεις:
  - Η ετικέτα `private` μπορεί να παραληφθεί
  - Αν πριν τη δήλωση ενός μέλους δεν αναφέρεται ρητά η προσβασιμότητα τότε θεωρείται `private`
  - Υπάρχει ένα ακόμη επίπεδο πρόσβασης (`protected`)
  - Τα πεδία συνήθως είναι ιδιωτικά, ενώ οι κατασκευαστές και οι μέθοδοι δημόσιες

Η ανατομία μίας κλάσης

- Το σώμα της κλάσης χωρίζεται σε δύο μέρη:
  - ιδιωτικό (private) Παρέχει τη διεπαφή της κλάσης
  - δημόσιο (public) Λεπτομέρειες της υλοποίησης
- Τα δημόσια μέλη αποτελούν τη διασύνδεση για τα αντικείμενα της κλάσης
- Τα ιδιωτικά μέλη είναι προσβάσιμα μόνο από συναστήσεις-μέλη της κλάσης

```
class ClassName {

private:
    Ιδιωτικά πεδία
    Ιδιωτικοί κατασκευαστές
    Ιδιωτικές μέθοδοι

public:
    Δημόσια πεδία
    Δημόσιοι κατασκευαστές
    Δημόσιες μέθοδοι

};
```

- Σημειώσεις:
  - Τα δεδομένα (πεδία) είναι ιδιωτικά, ώστε να είναι ασφαλή από χειραγώγηση
  - Οι μέθοδοι που εφαρμόζονται στα δεδομένα είναι δημόσιες, ώστε να ναι δυνατή η πρόσβαση τους εκτός κλάσης
  - Δεν υπάρχει κανόνας που να λέει ότι τα δεδομένα πρέπει να είναι ιδιωτικά και οι μέθοδοι δημόσιες
  - Σε ορισμένες περιπτώσεις ίσως χρειαστεί να χρησιμοποιήσετε ιδιωτικές μεθόδους και δημόσια δεδομένα

Επίπεδα προστασίας

- Όταν σχεδιάζουμε μια κλάση ορίζουμε το πως θα προσπελαύνονται τα μέλη δεδομένα (πεδία) και τα μέλη συναρτήσεις (μέθοδοι) της κλάσης
- Σε κάθε μέλος της κλάσης ανατίθεται ένα επίπεδο προστασίας:
  - **Δημόσια μέλη:** Πεδία και μέθοδοι που μπορούν να προσπελαστούν τόσο εκτός όσο και εντός των μελών συναρτήσεων της κλάσης
  - **Ιδιωτικά μέλη:** Πεδία και μέθοδοι που μπορούν να προσπελαστούν μόνο εντός των μεθόδων της κλάσης
- Πρόκειται για έναν μηχανισμό μέσω του οποίου υλοποιείται η **απόκρυψη πληροφορίας**

```
class Complex {
private:
  double real; // Real part
  double imag; // Imaginary part

public:
  Complex(double r, double i) {
   real = r;
   imag = i;
  }
  double getRealPart() {
    return real;
  }
  double getImaginaryPart() {
    return imag;
  }
  ..
};
```

```
int main() {
  Complex c(1,2);
  cout << c.real << endl;//error
  cout << c.getRealPart() << endl;
}
```

Part01/Complex.cpp

Τα πεδία είναι συνήθως ιδιωτικά: Γιατί;

- Επεξήγηση με ένα παράδειγμα:
  - Υποθέστε ότι τα πεδία `numerator` και `denominator` της κλάσης `Fraction` είναι δημόσια.
  - Η απευθείας πρόσβαση στο πεδίο `denominator` οδήγησε στην δημιουργία ενός κλάσματος με μηδενικό παρανομαστή.
  - Αυτό θα μπορούσε να είχε αποφευχθεί αν είχε χρησιμοποιηθεί η μέθοδος `setDenominator()`

```
//Class Fraction
class Fraction {
public:
  int numerator;
  int denominator;
  ...
  void setDenominator (int d) {
    if (d!=0)
      denominator = d;
  }
  double getValue() {
    return numerator/denominator;
  }
};
```

```
int main() {
  Fraction f;
  f.numerator = 1;
  f.denominator = 0;
  // This creates an error
  cout << f.getValue();
  return 0;
}
```

Part01/Fraction.cpp

Μέρος 2°:
Δήλωση και λεπτομέρειες υλοποίησης

Μια κλάση

- Μια κλάση που υλοποιεί ένα μετρητή χρόνου
  - Ώρες
  - Λεπτά
  - Δευτερόλεπτα

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

class Timer {
private:
  int hours, minutes, seconds;
public:
  /**
   * Construct a timer object initialized to 0:00:00
   */
  Timer(): hours(0), minutes(0), seconds(0) {
  }

  /**
   * Return the current time of this timer.
   */
  string getTime() {
    return to_string(hours)   + ":" +
           to_string(minutes) + ":" +
           to_string(seconds);
  }
  ...
};
```

Part01/Timer.cpp

Το όνομα της κλάσης

- Το όνομα της κλάσης είναι ένας προσδιοριστής:
  - ξεκινά με αλφαβητικό χαρακτήρα και
  - ακολουθεί οποιοσδήποτε συνδυασμός αλφαβητικών, αριθμητικών χαρακτήρων και του '_' [underscore]
- Δεν επιτρέπεται η χρήση λέξεων κλειδιών ως αναγνωριστικών
- **Συμβάσεις:**
  - Το όνομα της κλάσης αρχίζει με κεφαλαίο γράμμα
  - Κάθε επόμενο από το πρώτο συνθετικό ξεκινά με κεφαλαίο γράμμα π.χ., ComplexNumber
- Οι σταθερές πρέπει να είναι με κεφαλαία π.χ., Math.PI

```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

Part01/Timer.cpp

Το όνομα της κλάσης

- Το όνομα της κλάσης είναι ένας προσδιοριστής:
  - ξεκινά με αλφαβητικό χαρακτήρα και
  - ακολουθεί οποιοσδήποτε συνδυασμός αλφαβητικών, αριθμητικών χαρακτήρων και του '_' [underscore]
- Δεν επιτρέπεται η χρήση λέξεων κλειδιών ως αναγνωριστικών
- Η C++ είναι case-sensitive `Result` ≠ `result` ≠ `RESULT`

έγκυροι προσδιοριστές:

`number`
`x98`
`howMany`
`NUMBER_OF_POINTS`

λανθασμένοι:

`number of points`
`99`
`birth.year`
`ARRAY-SIZE`

```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

Part01/Timer.cpp

Πεδία (ή δεδομένα-μέλη)

- **Δήλωση πεδίου:** περιλαμβάνει τον τύπο και το όνομα του
  - το όνομα είναι ένα προσδιοριστής
  - ο τύπος καθορίζει την τιμή που μπορεί να λάβει
    π.χ., ακέραιος (`short`, `int`) ή δεκαδικός (`double`, `long`)

- **Σύμβαση:** Το όνομα ενός πεδίου αρχίζει με πεζό γράμμα

- Τα ονόματα των πεδίων πρέπει να «έχουν νόημα»
  - καλά: `yearOfBirth`, `numberOfSeats`, `totalTime`.
  - άσχημα: `yb`, `s`, `number`

```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

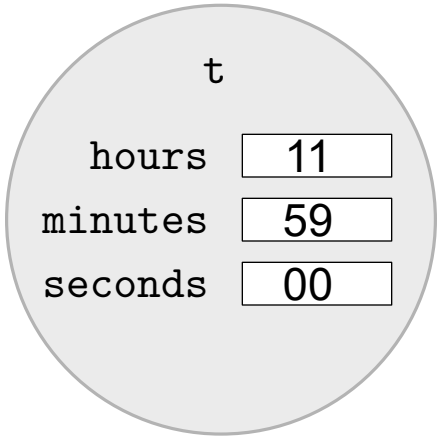
Part01/Timer.cpp

Πεδία (ή δεδομένα-μέλη)

- **Δήλωση πεδίου:** περιλαμβάνει τον τύπο και το όνομα του
  - το όνομα είναι ένα προσδιοριστής
  - ο τύπος καθορίζει την τιμή που μπορεί να λάβει
    π.χ., ακέραιος (`short`, `int`) ή δεκαδικός (`double`, `long`)

- **Σύμβαση:** Το όνομα ενός πεδίου αρχίζει με πεζό γράμμα

```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```



Οπτικοποίηση των πεδίων ενός αντικειμένου τύπου Timer

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

Part01/Timer.cpp

Μέθοδοι (ή συναρτήσεις-μέλη)

- **Δήλωση μιας μεθόδου:** περιλαμβάνει τον τύπο αποτελέσματος, το όνομα και τις παραμέτρους
  - ο τύπος αποτελέσματος καθορίζει τον τύπο της τιμής που επιστρέφει
    π.χ., ακέραιος (`short`, `int`) ή δεκαδικός (`double`) ή τίποτα (`void`)
  - το όνομα είναι ένα προσδιοριστής
  - οι παράμετροι έχουν τύπο και όνομα και χωρίζονται με κόμμα (αν είναι ≥ 1)
- Μια μέθοδος έχει μόνο ένα τύπο αποτελέσματος
- Το **“void”** είναι ένας ειδικός τύπος που υποδηλώνει ότι τίποτε δεν επιστρέφεται.
- **Σύμβαση:** Το όνομα μιας μεθόδου αρχίζει με πεζό γράμμα

```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

Part01/Timer.cpp

Μέθοδοι (ή συναρτήσεις-μέλη)

- **Δήλωση μιας μεθόδου:** περιλαμβάνει τον τύπο αποτελέσματος, το όνομα και τις παραμέτρους
- **Υλοποίηση ή σώμα μιας μεθόδου:**
  - είναι μια ακολουθία εντολών, που εκτελούνται σειριακά
  - η C++ υποστηρίζει δύο τύπους υλοποιήσεων

    Μέσα στη δήλωση της κλάσης

    Μετά τη δήλωση της κλάσης → ακολουθεί παράδειγμα

```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

Part01/Timer.cpp

Μέθοδοι (ή συναρτήσεις-μέλη)

- **Δήλωση μιας μεθόδου:** περιλαμβάνει τον τύπο αποτελέσματος, το όνομα και τις παραμέτρους
- **Υλοποίηση ή σώμα μιας μεθόδου:**
  - είναι μια ακολουθία εντολών, που εκτελούνται σειριακά
- **Κλήση μιας μεθόδου:**

  όνομαΑντικειμένου.όνομαΜεθόδου(παράμετροι)

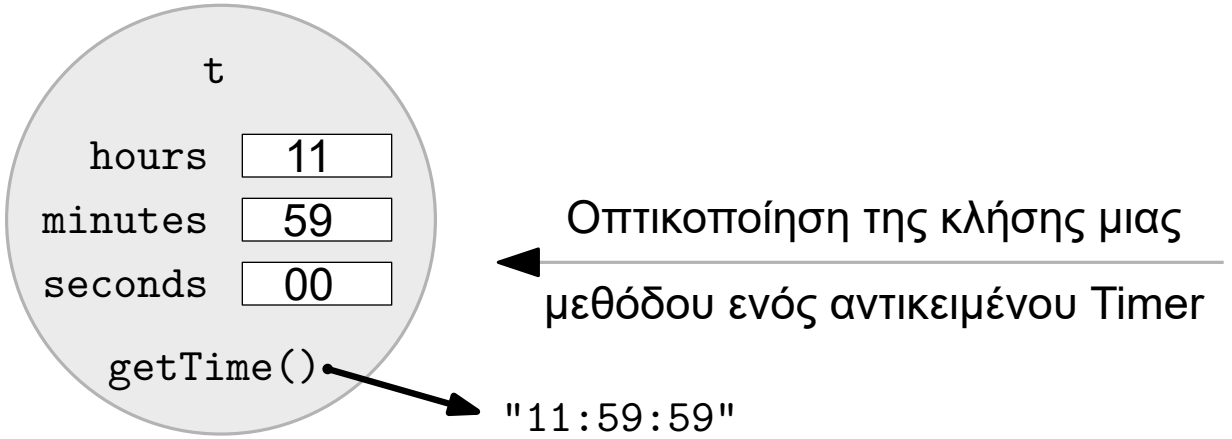
```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

Part01/Timer.cpp

Μέθοδοι (ή συναρτήσεις-μέλη)

- **Δήλωση μιας μεθόδου:** περιλαμβάνει τον τύπο αποτελέσματος, το όνομα και τις παραμέτρους
- **Υλοποίηση ή σώμα μιας μεθόδου:**
  - είναι μια ακολουθία εντολών, που εκτελούνται σειριακά
- **Κλήση μιας μεθόδου:**
  όνομαΑντικειμένου.όνομαΜεθόδου(παράμετροι)




```
class Timer {
private:
  int hours, minutes, seconds;
public:
  Timer() {
    hours = 0;
    minutes = 0;
    seconds = 0;
  }
  int getHours() {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  ...
};
```

```
int main() {
  Timer t;
  t.setHours(11);
  t.setMinutes(59);
  cout << t.getTime() << endl;
  return 0;
}
```

Part01/Timer.cpp

Περισσότερα για τις μεθόδους

- Το σώμα της μεθόδου αποτελείται από εντολές:
  - καταχώρησης [assignment]
  - επιστροφής αποτελέσματος [return]
  - κλήσεις μεθόδων
- Κάθε εντολή τερματίζεται με ένα ερωτηματικό «;»
- Η εντολή “return” επιστρέφει μια τιμή και τερματίζει την εκτέλεση της μεθόδου
- Ο τύπος της επιστρεφόμενης τιμής πρέπει να είναι ίδιος με τον τύπο-αποτελέσματος της μεθόδου.

```
class Cuboid {
private:
  int width, height, depth;

public:
  Cuboid(int w, int h, int d) {
    set(w, h, d);
  }

  // Computes the surface area
  int getSurfaceArea() {
    int front = height * width;
    int side = height * depth;
    int top = width * depth;
    return 2 * (front+side+top);
  }

  // Sets dimensions
  void set(int w, int h, int d) {
    width = w;
    height = h;
    depth = d;
  }
  ...
};
```

Part01/Cuboid.cpp

Περισσότερα για τις μεθόδους

- Η λίστα των παραμέτρων αποτελείται από ορισμούς παραμέτρων που χωρίζονται μεταξύ τους με κόμμα «,»
- Οι μέθοδοι μπορεί να χρησιμοποιούν τοπικές μεταβλητές στην υλοποίηση τους
- Η χρήση τοπικών μεταβλητών πρέπει να περιλαμβάνει τη δήλωση (τύπο, όνομα) και την αρχικοποίηση τους

```
class Cuboid {
private:
  int width, height, depth;

public:
  Cuboid(int w, int h, int d) {
    set(w, h, d);
  }

  // Computes the surface area
  int getSurfaceArea() {
    int front = height * width;
    int side = height * depth;
    int top = width * depth;
    return 2 * (front+side+top);
  }

  // Sets dimensions
  void set(int w, int h, int d) {
    width = w;
    height = h;
    depth = d;
  }
  ...
};
```

Part01/Cuboid.cpp

Περισσότερα για τις μεθόδους

- Η λίστα των παραμέτρων αποτελείται από ορισμούς παραμέτρων που χωρίζονται μεταξύ τους με κόμμα «,»
- Οι μέθοδοι μπορεί να χρησιμοποιούν τοπικές μεταβλητές στην υλοποίηση τους
- Η χρήση τοπικών μεταβλητών πρέπει να περιλαμβάνει τη δήλωση (τύπο, όνομα) και την αρχικοποίηση τους
  - Δήλωση:

```
int side;
string name;
Person father;
```

  - Δήλωση με αρχικοποίηση:

```
int side = 0;
string name = "Alice";
Person father("Bob", 65);
```

```
class Cuboid {
private:
  int width, height, depth;

public:
  Cuboid(int w, int h, int d) {
    set(w, h, d);
  }

  // Computes the surface area
  int getSurfaceArea() {
    int front = height * width;
    int side = height * depth;
    int top = width * depth;
    return 2 * (front+side+top);
  }

  // Sets dimensions
  void set(int w, int h, int d) {
    width = w;
    height = h;
    depth = d;
  }
  ...
};
```

Part01/Cuboid.cpp

Περισσότερα για τις μεθόδους

- Η λίστα των παραμέτρων αποτελείται από ορισμούς παραμέτρων που χωρίζονται μεταξύ τους με κόμμα «,»
- Οι μέθοδοι μπορεί να χρησιμοποιούν τοπικές μεταβλητές στην υλοποίηση τους
- Η χρήση τοπικών μεταβλητών πρέπει να περιλαμβάνει τη δήλωση (τύπο, όνομα) και την αρχικοποίηση τους
  - Σφάλμα (χρήση χωρίς αρχικοποίηση):

```
int side;
int max;
max = 2 * side;
```

```
class Cuboid {
private:
  int width, height, depth;

public:
  Cuboid(int w, int h, int d) {
    set(w, h, d);
  }

  // Computes the surface area
  int getSurfaceArea() {
    int front = height * width;
    int side = height * depth;
    int top = width * depth;
    return 2 * (front+side+top);
  }

  // Sets dimensions
  void set(int w, int h, int d) {
    width = w;
    height = h;
    depth = d;
  }
  ...
};
```

Part01/Cuboid.cpp

Κατασκευαστές (Constructors)

- Ο κατασκευαστής είναι μια ειδική μέθοδος που εκτελείται όταν δημιουργείται ένα αντικείμενο
- Σκοπός του είναι η αρχικοποίηση των πεδίων του αντικειμένου σε μια έγκυρη κατάσταση
- Ένας κατασκευαστής δεν έχει τύπο-αποτελέσματος
- Το όνομα της μεθόδου-κατασκευαστή είναι το ίδιο με αυτό της κλάσης
- Ο όρος **προκαθορισμένος κατασκευαστής** αναφέρεται σε έναν κατασκευαστή χωρίς παραμέτρους
- Κάθε κλάση **θα πρέπει** να έχει έναν κατασκευαστή
  - Αν δεν γραφεί από τον προγραμματιστή τότε ένας κενός προκαθορισμένος κατασκευαστής δημιουργείται από τη γλώσσα

```
class Timer
{
private:
  int hours, minutes, seconds;

public:
  /**
   * Default constructor
   * Construct a timer at 0:0:0
   */
  Timer() {
    hours   = 0;
    minutes = 0;
    seconds = 0;
  }
  /**
   * Construct a timer at h:m:s
   */
  Timer(int h, int m, int s) :
        hours(h),
        minutes(m),
        seconds(s) {}
  ...
};
```

Part01/Timer.cpp

Κατηγορίες μεθόδων

- Κατασκευαστής [constructor]
  - δημιουργεί και αρχικοποιεί το αντικείμενο.
- Μέθοδος προσπέλασης [accessor / selector / geter]
  - Επιστρέφει μέρος των δεδομένων ενός αντικειμένου.
  - Δεν τροποποιεί τα δεδομένα του αντικειμένου.
  - Δηλώνεται ως `const`.
- Μέθοδος μετάλλαξης [mutators / seter]
  - Αλλάζει την κατάσταση (δεδομένα) ενός αντικειμένου.
  - Δεν μπορεί να είναι `const`.

```
class Timer
{
private:
  int hours, minutes, seconds;

public:
  Timer() {
    hours   = 0;
    seconds = 0;
    minutes = 0;
  }
  int getHours() const {
    return hours;
  }
  void setHours(int h) {
    hours = h % 12;
  }
  string getTime() const {
    return to_string(hours) + ":"
       + to_string(minutes) + ":"
       + to_string(seconds);
  }
  ...
};
```

Part01/Timer.cpp

Υλοποίηση μεθόδων εκτός της κλάσης

- Η αναφορά σε μια μέθοδο γίνεται ως εξής:
  όνομαΚλάσης::όνομαΜεθόδου(παράμετροι)
- Είναι ένας τρόπος ορισμού ποια κλάση σχετιζεται με ποιες μεθόδους
- Ο τελεστής :: ονομάζεται τελεστής προσδιορισμού εμβέλειας [scope resolution operator]
- **Σύμβαση:** Αν το σώμα μιας μεθόδου είναι:
  - μικρό, τότε η υλοποιήση γίνεται μέσα στη δήλωση της κλάσης
  - μεγάλο, τότε η υλοποιήση γίνεται μετά τη δήλωση της κλάσης

```
class Circle {
private:
  //Private members
  int radius;
public:
  //Public members
  Circle();
  double area();
  double circumference();
  int getRadius();
  void setRadius(int r);
};
```

```
//Methods implementation
Circle::Circle() : radius(1) {
  // equivalently: radius = 1;
}
double Circle::area() {
  return 3.14*radius*radius;
}
double Circle::circumference() {
  return 2*3.14*radius;
}
...
```

Part01/Circle.cpp

Μέρος 3°:
Δημιουργία αντικειμένων

Δημιουργία αντικειμένων

- **Υπενθύμιση:** Μια κλάση είναι ένας οριζόμενος από τον χρήστη τύπος
- Αντικείμενα δημιουργούνται με βάση τις κλάσεις
  - Οι κλάσεις ένα είδος «βιομηχανίας» αντικειμένων
  - Τα αντικείμενα είναι στιγμιότυπα των κλάσεων
- Πολλά αντικείμενα μπορεί να δημιουργηθούν βασιζόμενα στην ίδια κλάση
- Η δημιουργία αντικειμένων γίνεται ως εξής:

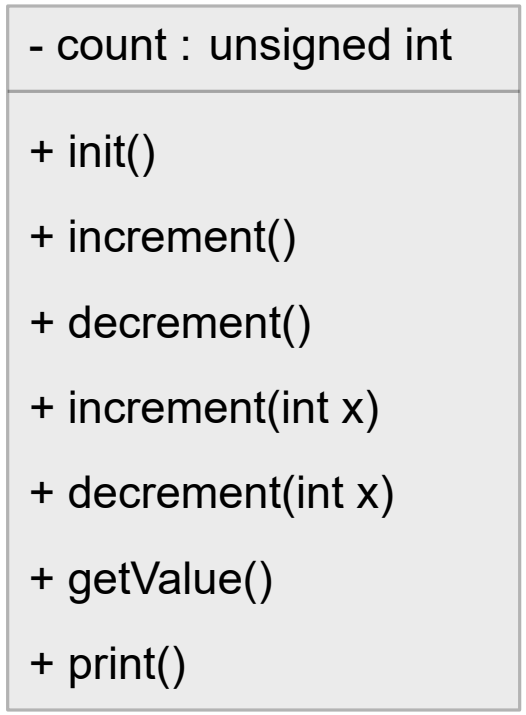
  ΌνομαΚλάσης όνομαΑντικειμένου(παράμετροι)

```
class Point {
private:
  int x, y;
public:
  ...
};
class LineSegment {
private:
  Point start, end;
public:
   ...
};

int main() {
  // Create two points
  Point p1();
  Point p2(0,1);

  // Create two line segments
  LineSegment s1(p1, p2);
  LineSegment s2(0,1,0,1);

  // Print the length of s1
  cout << s1.getLength() << endl;
}
```

Part01/Point.cpp, Part01/LineSegment.cpp

Επίδειξη: ανάπτυξη μιας κλάσης

- Κλάση Counter (μετρητής)



Part01/Counter.cpp

Μέρος 4°:

Σχόλια και διαμόρφωση κώδικα

Σχόλια

- Τα σχόλια προστίθενται για να κάνουν κατανοητό τον κώδικα
- Αποτελούν σημειώσεις που μπορεί να είναι σημαντικές για τον προγραμματιστή αλλά αγνοούνται από το μεταγλωττιστή
- Σχόλιο μιας γραμμής:

```
// This is a single line comment
```

- Σχολιο πολλαπλών γραμμών:

```
/**
 * This is a multi-line comment
 * This is its second line
 */
```

```
//Class Fraction
#include<iostream>
using namespace std;

class Fraction {
private:
  int numerator;
  int denominator;

public:
  /**
   * Default constructor
   */
  Fraction() {
    numerator = 0;
    denominator = 1;
  }
  /**
   * Return the fraction's value
   */
  double getValue() {
    return numerator/denominator;
  }
  ...
};
```

Part01/Fraction.cpp

Διαμόρφωση κώδικα

- Οι οδηγίες διαμόρφωσης κειμένου [style guidelines] περιγράφουν τρόπους διάταξης και τεκμηρίωσης του πηγαίου κώδικα
- Αποσκοπούν στο να κάνουν την κατανόηση του κώδικα ευκολότερη
- Αφορούν:
  - σχόλια
  - διάταξη (ευθυγράμμιση κειμένου, «κενά»)
  - ονόματα μεταβλητών / κλάσεων / μεθόδων / …
- Τα σύγχρονα περιβάλλοντα ανάπτυξης υποστηρίζουν αυτόματη διαμόρφωση του πηγαίου κώδικα

```cpp
//Class Fraction
#include<iostream>
using namespace std;

class Fraction {
private:
  int numerator;
  int denominator;

public:
  /**
   * Default constructor
   */
  Fraction() {
    numerator = 0;
    denominator = 1;
  }
  /**
   * Return the fraction's value
   */
  double getValue() {
    return numerator/denominator;
  }
  ...
};
```

Part01/Fraction.cpp

Μέρος 5°:

Προχωρημένα θέματα υλοποίησης

Αντικείμενα ως παράμετροι μεθόδων

- Αντικείμενα μπορούν να αποτελούν παραμέτρους μεθόδων (όπως οι βασικοί τύποι δεδομένων)
- Για την προσπέλαση της κατάστασης τους χρησιμοποιούμε τις μεθόδους τους
- Όμοια μια μέθοδος μπορεί να επιστρέφει αντικείμενο

```
class Distance
{
private:
  unsigned int meters, centimeters;

public:
  void increaseBy(Distance d) {
    meters += d.getMeters();
    centimeters += d.getCentimeters();
    if (centimeters > 100) {
      meters += 1;
      centimeters -= 100;
    }
  }
  ...
};
```

```
int main() {
  Distance d1(2, 40);
  Distance d2(1, 20);
  d1.increaseBy(d2);
  cout << "d1=" << d1.getMeters() <<
  "," << d1.getCentimeters() << endl;
}
```

Part01/Distance.cpp

Στατικά πεδία

- Τα στατικά πεδία δεδομένων ανήκουν σε μια κλάση και όχι σε ένα αντικείμενο
- Τα στατικά πεδία είναι κοινόχρηστα από όλα τα αντικείμενα μίας κλάσης – κάθε στιγμιότυπο [instance] της ίδιας κλάσης χρησιμοποιεί τα ίδια στατικά πεδία
- Υπάρχει μόνο ένα αντίγραφο ενός στατικού πεδίου μίας κλάσης ανεξάρτητα από τον αριθμό των στιγμιότυπων της κλάσης που έχουν δημιουργηθεί

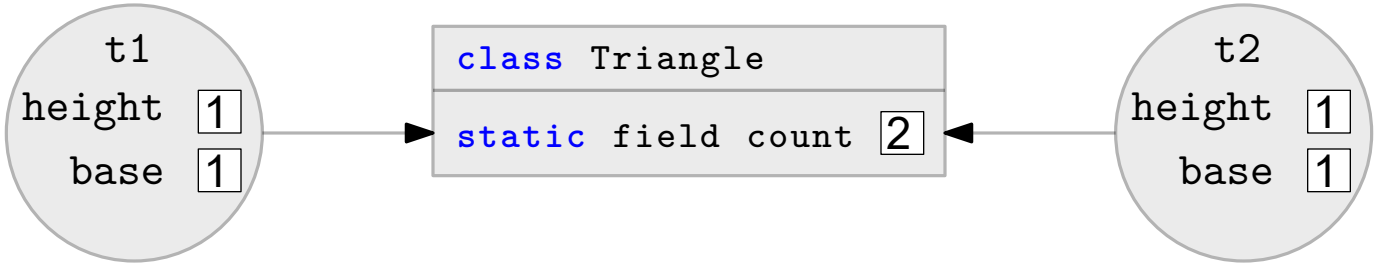
```
class Triangle {
private:
  int height, base;
  static int count;
public:
  Triangle(): height(1), base(1) {
    count++;
  }
  int getCount() {
    return count;
  }
  ...
};
// Initialize static member
int Triangle::count = 0;
```

```
int main() {
  //Instantiate two objects and
  //output its number
  Triangle t1();
  Triangle t2();
  cout << t1.getCount();
}
```

Part01/Triangle.cpp

Στατικά πεδία

- Τα στατικά πεδία δεδομένων ανήκουν σε μια κλάση και όχι σε ένα αντικείμενο
- Τα στατικά πεδία είναι κοινόχρηστα από όλα τα αντικείμενα μίας κλάσης – κάθε στιγμιότυπο [instance] της ίδιας κλάσης χρησιμοποιεί τα ίδια στατικά πεδία
- Υπάρχει μόνο ένα αντίγραφο ενός στατικού πεδίου μίας κλάσης ανεξάρτητα από τον αριθμό των στιγμιότυπων της κλάσης που έχουν δημιουργηθεί




```
class Triangle {
private:
  int height, base;
  static int count;
public:
  Triangle(): height(1), base(1) {
    count++;
  }
  int getCount() {
    return count;
  }
  ...
};
// Initialize static member
int Triangle::count = 0;
```

```
int main() {
  //Instantiate two objects and
  //output its number
  Triangle t1();
  Triangle t2();
  cout << t1.getCount();
}
```

Part01/Triangle.cpp

Στατικές μέθοδοι

- Οι στατικές μέθοδοι «ανήκουν» σε μια κλάση και όχι σε ένα αντικείμενο (μέθοδοι κλάσης [class methods])
- Δεν απαιτείται η ύπαρξη αντικείμενου για την κλήση τους (καλούνται «επί της κλάσης»)
- Οι στατικές μέθοδοι δεν μπορούν να έχουν πρόσβαση σε μη στατικά πεδία μιας κλάσης

```
class Triangle {
private:
  int height, base;
  static int count;
public:
  Triangle(): height(1), base(1) {
    count++;
  }
  static int getCount() {
    return count;
  }
  ...
};
// Initialize static member
int Triangle::count = 0;
```

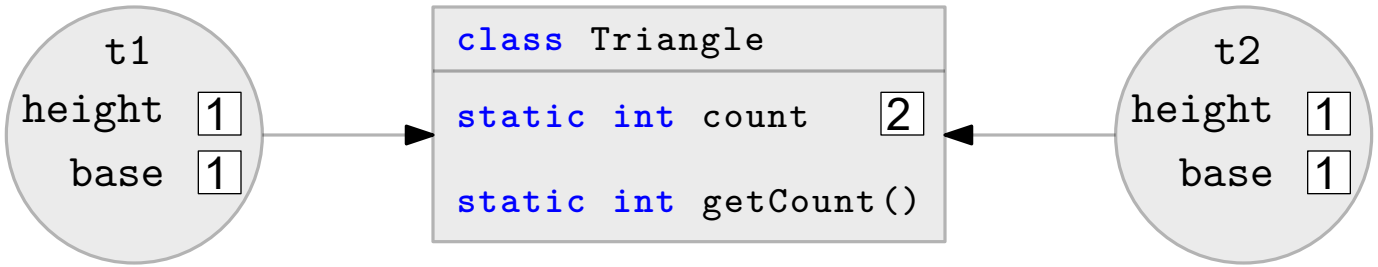
```
int main() {
  //Instantiate two objects and
  //output its number
  Triangle t1();
  Triangle t2();
  cout << Triange::getCount();
}
```

Part01/Triangle.cpp

Στατικές μέθοδοι

- Οι στατικές μέθοδοι «ανήκουν» σε μια κλάση και όχι σε ένα αντικείμενο (μέθοδοι κλάσης [class methods])
- Δεν απαιτείται η ύπαρξη αντικείμενου για την κλήση τους (καλούνται «επί της κλάσης»)
- Οι στατικές μέθοδοι δεν μπορούν να έχουν πρόσβαση σε μη στατικά πεδία μιας κλάσης

```
class Triangle {
private:
  int height, base;
  static int count;
public:
  Triangle(): height(1), base(1) {
    count++;
  }
  static int getCount() {
    return count;
  }
  ...
};
// Initialize static member
int Triangle::count = 0;
```




```
int main() {
  //Instantiate two objects and
  //output its number
  Triangle t1();
  Triangle t2();
  cout << Triange::getCount();
}
```

Part01/Triangle.cpp

Καταστοφείς

- Κάθε κλάση διαθέτει και μια ειδική συνάρτηση που ονομάζεται καταστροφέας [destructor]
- Ο καταστροφέας μοιάζει με τον προκαθορισμένο κατασκευαστή αλλά φέρει το σύμβολο ∼ στο όνομά του
  - Ο καταστροφέας της κλάσης `Person` είναι: `~Person()`
- Όπως και με τους κατασκευαστές, οι καταστροφείς καλούνται αυτόματα (όχι ρητά) πριν το αντικείμενο αποδεσμευτεί από το σύστημα
- Η τυπική εργασία ενός καταστροφέα είναι να πραγματοποιεί όποιες εργασίες αποδέσμευσης πόρων (συνήθως μνήμης) απαιτούνται, πριν το αντικείμενο αποδεσμευτεί

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

class Student {
private:
  string name;
  int age;

public:
  Student(string n, int a):
          name(n), age(a) {
    cout<<"Construct "<<name;
  }
  ~Student(){
    cout<<"Destruct "<<name;
  }
};
```

```cpp
int main() {
  Student s1("Bob", 30);
  Student s2("Alice", 29);
  return 0;
}
```

Part01/Student.cpp

const

- Γενικά, η δεσμευμένη λέξη `const` δηλώνει ότι το αναγνωριστικό στο οποίο εφαρμόζεται δεν δύναται να τροποποιηθεί.
- const μεταβλητές είναι εκείνες οι μεταβλητές που έχουν μόνο μια τιμή (αυτή με την οποία αρχικοποιήθηκαν)
- const αντικείμενα δεν μπορούν να τροποποιηθούν μετά την αρχικοποίηση τους
  - Ο μεταγλωττιστής επιβάλλει ότι ένα const αντικείμενο μπορεί να καλεί μόνο const μεθόδους, οι οποίες δεν τροποποιούν τις τιμές των πεδίων

```
int main() {
  const int x = 5;
  x=2;
  return 0;
}
```

```
const int SIZE = 10;
const double PI = 3.1415;
const Fraction ZERO(0,1);
const Fraction FIXED(3,4);
```

```
class Circle {
  int area() const {
    return 3.14*radius*radius;
  }
  void setRadius(int r) {
    radius = r;
  }
  ...
};
int main() {
  const Circle UNIT_CIRCLE(1);
  cout<<UNIT_CIRCE.area();
  UNIT_CIRCLE.setRadius(2);
}
```

const

- const πεδία πρέπει απαραίτητα να αρχικοποιούνται στην λίστα αρχικοποίησης (και όχι στο σώμα) του κατασκευαστή
  - η τιμή τους δεν μπορεί να τροποποιηθεί μετά την αρχικοποίηση του αντικειμένου
  - γενικά, η χρήση τους δεν είναι σύνηθης
- const μέθοδοι δεν μπορούν να τροποποιήσουν τις τιμές των πεδίων ενός αντικειμένου
- const παράμετροι δεν μπορούν να τροποποιηθούν στην αντίστοιχη μέθοδο/συνάρτηση

```
class DataEntry
{
private:
  const int id;
  string value;
  static int count;

public:
  DataEntry(string s):id(count){
    value = s;
    count++;
  }
  int getId() const {
    return id;
  }
  string getValue() const {
    return value;
  }
  void setValue(const string s){
    value = s;
  }
};
int DataEntry::count = 0;
```

Part01/DataEntry.cpp

Μέρος 6°:

Projects πολλαπλών αρχείων

Projects πολλαπλών αρχείων

- Αν και είναι δυνατόν οποιοδήποτε πρόγραμμα να γραφεί σε ένα μόνο αρχείο, συχνά τα προγράμματα διαμερίζονται σε πολλά αρχεία:
  - **Αρχείο επικεφαλίδας (header):** περιέχει τη δήλωση της κλάσης, συνήθως έχει ως επέκταση hpp π.χ. Circle.hpp
  - **Αρχείο υλοποίησης:** περιέχει τον ορισμό της κλάσης, δηλαδή της υλοποίησης των μεθόδων της κλάσης, συνήθως έχει ως επέκταση cpp π.χ. Circle.cpp
  - **Αρχείο οδηγός:** περιέχει την main() και πιθανώς άλλες συναρτήσεις που χρησιμοποιούνται στο πρόγραμμα, συνήθως έχει ως επέκταση cpp π.χ. main.cpp
- Για την μεταγλώττιση ενός project πολλαπλών αρχείων θα πρέπει να συμπεριληφθούν όλα τα σχετικά αρχεία:

```
$ g++ Circle.hpp Circle.cpp main.cpp -o main.exe
```

Projects πολλαπλών αρχείων

- Λόγοι για τους οποίους χρησιμοποιούνται πολλαπλά αρχεία:
  - Επιτρέπει να γίνονται αλλαγές στην υλοποίηση των κλάσεων χωρίς να επηρεάζεται η διεπαφή της κλάσης.
  - Επιτρέπει σε μια κλάση να χρησιμοποιείται από πολλά προγράμματα οδηγούς.
  - Αυξάνει τις ευκαιρίες για επαναχρησιμοποίηση κώδικα.
  - Ο κώδικας γίνεται περισσότερο τμηματικός (modular)
  - Στην πράξη, πολλές φορές, δεν είναι δυνατόν να γραφεί κώδικας για ένα μεγάλο έργο λογισμικού σε ένα μόνο αρχείο.

Επίδειξη: ανάπτυξη μιας κλάσης

- Κλάση Vector (διάνυσμα)

`Part01/Vector/Vector.cpp`